VERIFIABLE CONTROL SYNTHESIS FOR ROBOTIC SWARMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Ji Chen

May 2022

VERIFIABLE CONTROL SYNTHESIS FOR ROBOTIC SWARMS

Ji Chen, Ph.D.

Cornell University 2022

Swarm robotics is an active research area where large groups of simple robots are developed to perform complex tasks. It has various potential applications such as surveillance, warehouse logistics, and collective construction, because of its large sensing capability, fault tolerance, and collaboration skills that are impossible to a single robot. However, controlling swarm robots to perform high-level tasks automatically and verifiably remains challenging due to the large state space. In this work, I present the verifiable control framework for swarm robots from high-level specifications in different combinations of 4 dimensions: symbolic/continuous levels, centralized/decentralized methods, navigation/formation-based specifications, and non-reactive/reactive tasks.

The first contribution of the work is a top-down control framework for non-reactive navigation swarm tasks. At the symbolic level, I propose an integer programming-based method for executing a centralized symbolic plan, and compare it with existing decentralized execution in terms of task efficiency, failure resilience, and computational complexity. At the continuous level, I propose a control barrier function based method which guarantees correct (satisfy all specifications) and safe (no collision) transitions when robots execute the high-level tasks. The trade-offs between centralized and decentralized methods provide guidance to swarm controls to achieve different purposes under different restrictions.

The second contribution is a novel abstraction and grammar for location and formation-based swarm specifications. With the proposed abstractions, we developed a centralized control synthesis approach which guarantees that the specified swarm be-

haviors will be satisfied if feasible. This contribution expands the task space of swarms, and gives insights into controlling a large fleet of autonomous robots to perform complex tasks which require composition of behaviors and coordination of different sub-swarms.

The third contribution is an extension to the second, which automatically synthesizes controls for swarms to achieve reactive formation tasks in a decentralized manner. This work utilizes an integer programming-based method to calculate constraints on sub-swarm sizes given the reactive finite state machine, to ensure a priori feasibility of the task. In addition, a decentralized auction-based algorithm is developed for executing the reactive finite state machine, which satisfies the robot number constraints. Such decentralized frameworks increase the scalability of control synthesis and can be applied to swarm systems with a larger number of robots.

The last contribution of this thesis is an automatic approach to redistributing robots according to user requests during the task execution while maintaining the original specifications. The approach includes creating symbolic plans, limiting robot numbers given the constraints, and ensuring all robots to satisfy safety requirements of the given high-level tasks. This work further enhance the flexibility and reactivity of swarms performing high-level tasks.

BIOGRAPHICAL SKETCH

Ji Chen was born in Wuhan, China in 1994. He went to Tsinghua University in Beijing, China from 2012 to 2016. After he completed the Bachelor's degree in Mechanical Engineering there, he joined the Verifiable Robotics Research Group and started the graduate study at the Sibley School of Mechanical and Aerospace Engineering at Cornell University. His research focuses on developing verifiable control frameworks for robotic swarms to achieve safe and correct high-level behaviors.

For my parents.

ACKNOWLEDGEMENTS

First of all, I want to thank my parents for always supporting me unconditionally. Although I haven't met them in person for years due to COVID19, I can feel their love and consideration everyday from the other side of the earth.

I have spent fantastic five years at Cornell and so many people here have helped and inspired me during my graduate study. I give the most appreciation to my advisor Professor Hadas Kress-Gazit, for her patient guidance, valuable feedback and tremendous support. I learned a lot from her both on research and life. She is the most amazing advisor that I can hope for.

I would also like to thank my committee members Professor Kirstin Petersen and Professor Mark Campbell for their great suggestions to my research.

I also benefit a lot from interactions with other faculty members in MAE. Thanks to Professor Guy Hoffman for his deep understanding in HRI, to Professor Brian Kirby for the presentation skills, and to Professor Dmitry Savransky for discussions on dynamics problems. Of course, I want to thank Marcia for giving all the useful information and organizing the relaxing activities in the department.

In addition, I would like to thank everyone in VRRG and ASL for the joyful time we have spent in the past five years. Especially, I want to thank Adam and Thais for reading my papers, giving feedback to my presentations, and discussing on my research. Also, I want to thank Salar Moarref for the collaboration and his help during the first few semesters in my PhD.

In the end, I want to thank my buddies who spend plenty of time with me in the gym and on basketball court, which makes my PhD life much more fun.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Swarm robotics takes the inspiration from the self-organized social animals and studies the coordination and interaction of large numbers of simple robots, while achieving robust, scalable, and complex collective behaviors [13]. Researchers have developed various local interaction rules for individual robots to create the desired emergent behaviors, such as probabilistic aggregation [83], task division [46], and shape formation [87], etc. Such researches take the bottom-up design method, where local control protocols are developed through iterations. Another direction of designing controls for swarms is top-down, where high-level specifications are given and synthesis tools are used to obtain controls automatically with correctness guarantees [41, 43, 54]. In the thesis, I present my work on verifiable control from high-level specifications for swarm tasks, which contains three different control frameworks and they are introduced in the following three chapters.

First, in Chapter 2, I present a framework for creating provably-correct controllers for swarm navigation tasks from high-level specifications, and investigate the trade-offs between centralized and decentralized control. This part is based on my paper published in the International Conference on Autonomous Agents and Multiagent Systems in 2018 [17]. Specifically, I differentiate centralized and decentralized control at the symbolic and continuous levels. At the symbolic level, centralized or decentralized indicates whether robots share the same symbolic plan or each robot executes an individual plan. At the continuous level, centralized or decentralized control refers to the global or local information that the robots need. For the symbolic level, I first automatically synthesize a centralized global symbolic plan from high-level task specifications expressed in linear temporal logic (LTL). Then, I design an optimization-based dynamic assigning

method to execute the centralized symbolic plan directly. I compare the centralized optimal execution with the decentralized execution that we proposed in previous work in terms of efficiency, failure resilience and computational complexity. At the continuous level, I automatically compute control inputs using control barrier functions that execute the symbolic plans while ensuring collision avoidance, and describe methods for mitigating deadlocks that might occur. I compare the computation of the control inputs in centralized and decentralized control barrier functions in terms of computation complexity and robustness to possible failures. I demonstrate our approaches in MATLAB simulations and physical swarm platforms including Spheros and the Robotarium. I discuss the trade-offs between centralized and decentralized control at both the symbolic and continuous levels.

In Chapter 3, I introduce an abstraction that captures high-level formation and location-based swarm behaviors, and an automated control synthesis framework to generate correct-by-construction behaviors, which are based on my paper published in the International Conference on Intelligent Robots and Systems in 2020 [19]. The abstraction includes symbols representing both possible formations and physical locations in the workspace. Users can write linear temporal logic (LTL) specifications over the symbols to specify high-level tasks for the swarm. To satisfy a specification, I automatically synthesize a centralized symbolic plan, and environment and swarm-size-dependent motion controllers that are guaranteed to implement the symbolic transitions. In addition, using integer programming (IP), I assign robots to different sub-swarms to execute the synthesized symbolic plan. The framework gives insights into controlling a large fleet of autonomous robots to achieve complex tasks which require composition of behaviors at different locations and coordination among different groups of robots in a correct-by-construction way.

In Chapter 4, I present a distributed strategy for automatically synthesizing controls for robotic swarms such that they achieve reactive, high-level formation tasks. This work is based on my paper that will be published in the International Conference on Automation Science and Engineering in 2021. In the framework, a user specifies formation and location-based swarm tasks, which may include reactions to environmental events, using linear temporal logic. Then, I synthesize a centralized finite automaton that represents the symbolic behavior of the swarm. To execute the automaton, I develop an auction-based decentralized algorithm that assigns robots to different locations and formations using only information from neighboring robots. To guarantee that the swarm can achieve the specified high-level tasks, I use integer programming to obtain the maximum and minimum number of robots that need to be sent to different locations during each symbolic transition, and I incorporate the constraints on sub-swarm sizes into the auction-based assignment algorithm.

In chapter 5, I present a control framework for automatically redistributing robots based on user requests, received during execution, such that the original specification and the robot number constraints are satisfied, given a robot swarm performing a high-level task given in linear temporal logic. The approach is based on creating symbolic plans, limiting robot numbers given region constraints, and ensuring all robots maintain the safety requirements of the original task if possible; if not possible, we provide feedback to the user. I demonstrate the framework with simulated swarm robots and present quantitative analysis on the computation time. The approach increases the flexibility and reactivity of swarms performing high-level tasks while still guaranteeing the specification satisfaction. This is the first work to incorporate runtime changes to the specification while maintaining the original high-level specifications, in the context of robot swarms.

CHAPTER 2

CENTRALIZED AND DECENTRALIZED CONTROL OF ROBOTIC SWARMS FROM HIGH-LEVEL SPECIFICATIONS

2.1 Introduction

Swarm robotics is an active area of research due to their ability to distribute sensing and action [9], their fault tolerance to individual failures [90], and ability to achieve tasks that are difficult or impossible for a single robot [69]. Designing control schemes for swarm robots to perform the desired collective behaviors and achieve high-level tasks remains a challenging problem [14] due to the large state space. Research in control of swarm robots can be divided into two approaches: *centralized control*, where the full state of the robotic swarm is known and used in the control generation [53, 49], and *decentralized control*, where each robot makes decisions and computes controllers based on its local sensing and communication [32, 48].

In this chapter, we present and compare centralized and decentralized control schemes for swarm robots to achieve high-level tasks. The tasks we address are swarm robots visiting regions in a bounded workspace with the user-specified requirements expressed in linear temporal logic (LTL). As a motivating example, we consider a workspace consisting of rooms $A-F$ connected by a central region G, as shown in Fig. 2.1. In this dynamic environment, there are people (green triangles) walking around and static obstacle(s) (blue rectangles) which might block part of the entrance to a room. Additionally, there are walls (solid black lines) that bound the workspace and rooms. We assume that there is a group of robots of an unknown size (black circles) performing a high-level task which requires them to occupy A, B, and D simultaneously while always avoiding entering region C. We can abstract the system by using Boolean vari-

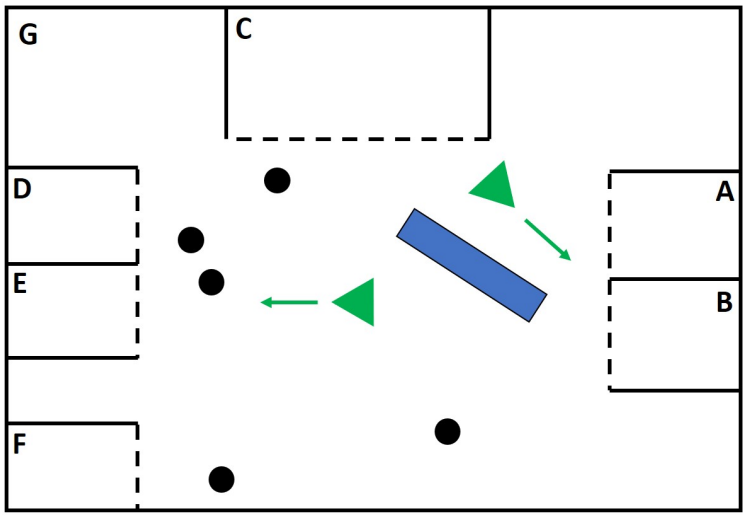


Figure 2.1: A swarm performing a room occupying task. Black circles represent robots, blue rectangles represent static obstacles, green triangles represent people (dynamic obstacles), and solid black lines represent walls.

ables to define occupancy of regions, and encode the high-level tasks by formulas of LTL [10]. In previous works [55] we developed methods to synthesize a symbolic plan (i.e., sequence of symbols) to achieve such a high-level task. In order to continuously execute the plan in a safe, correct and efficient manner, we must ensure that:

- robots do not collide with each other
- robots do not collide with obstacles, walls, or people
- robots do not enter another region when required to move between two regions (e.g., when a robot moves from G to A in Fig. 2.1, it should not pass through any regions other than A or G.)
- the resulting system is free of deadlocks, i.e., the robots must always be able to make progress toward their goals
- robots perform the task efficiently

- robots must be as resilient to unexpected failures as possible, e.g., if some robots lose connection and fail to move, the other robots can still complete the task.

To address these challenges, we approach the execution of symbolic plans in two ways, as shown in Fig. 2.2: *centralized* and *decentralized*. First, we synthesize a global symbolic plan. One way to implement the symbolic plan is to dynamically assign robots to different destinations during execution in a centralized manner. The other way is to use the framework in [55] to partition the global plan and obtain a set of decentralized symbolic plans associated with synchronization skeletons that send synchronization signals among sub-swarms (i.e., different sub-swarms need to wait for others at certain states in the symbolic plan). For example, if the centralized plan requires a swarm to occupy regions A, B, and D at the same time, robots need synchronization in the decentralized plans to guarantee the whole swarm indeed occupy A, B, and D simultaneously.

To ensure safety (i.e. collision-free motion), we leverage control barrier functions (CBFs) [89] to design control inputs for the robots at the continuous level. The control barrier function can be formulated in both centralized and decentralized manners. The centralized CBF considers the state space of the swarm system as the Cartesian product of all individual robots' state space, where the controller takes in the full state and returns control inputs (e.g., velocities for first-order systems) of all robots. In contrast, in the decentralized CBF, the computation is distributed and each robot takes in only local information (i.e., its own states and the states of other robots in the sensing range) and then computes the control input.

To validate the proposed swarm control framework, we synthesize controllers for MATLAB simulations and different physical swarm robotic systems - Sphero SPRK robots and Robotarium, a remotely accessible swarm testbed [65]. We automatically generate a centralized symbolic plan from the high-level specifications offline. Then, we

execute the centralized symbolic plan with a dynamic assigning method, and compare the centralized execution with decentralized execution in [17]. In addition, we compare the centralized and decentralized computation of CBF at the continuous level and discuss the trade-offs. In the following, we will use centralized and decentralized *symbolic plans* to represent controllers at the symbolic level, and centralized and decentralized *CBFs* to represent control at the continuous level.

This chapter is based on our previous work [17] on verifiable controller synthesis for swarms from high-level specifications. The contributions of this work are threefold: 1) an optimization-based dynamic assigning strategy that enables swarm robots to execute a centralized symbolic plan; 2) a comparison of the centralized and decentralized control at both the symbolic and continuous levels; 3) demonstrations of high-level swarm behaviors using both centralized and decentralized control on physical swarm systems to illustrate the differences.

The remainder of this chapter is structured as follows: In section 2.2, we discuss related work. In section 2.3, we briefly introduce the backgrounds of linear temporal logic (LTL) synthesis for swarm behaviors and control barrier functions for swarm control. In section 2.4, we present continuous control synthesis for collision avoidance, region invariance, and deadlock mitigation based on our previous work [17]. In section 2.5, we compare centralized and decentralized continuous control in the presence of individual robot failures. In section 2.6, we present an optimization-based dynamic assignment method to execute the centralized symbolic plan, and compare different behaviors of executing centralized and decentralized plans in both simulations and physical swarm systems. In section 2.7, we present a summary of our work.

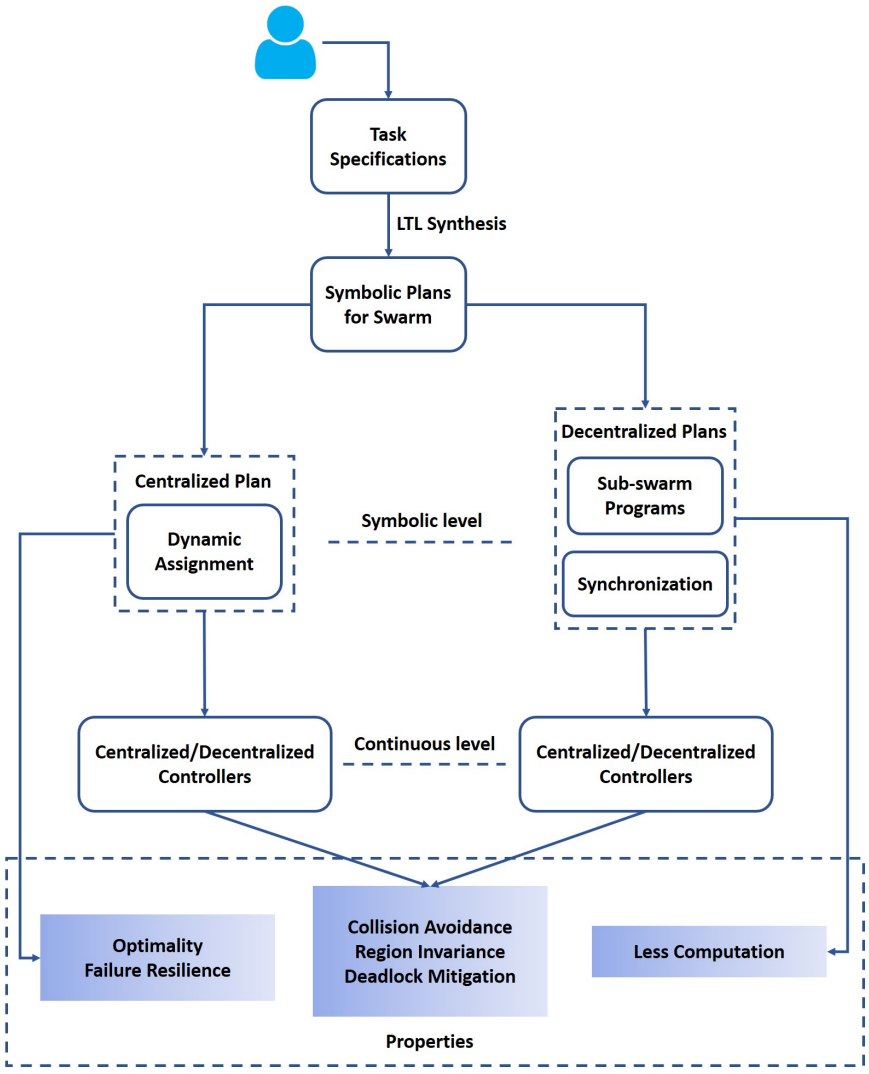


Figure 2.2: Centralized and decentralized swarm control from high-level specifications.

2.2 Related Work

Synthesis for multi-agent systems: In [41, 59], the authors synthesize centralized control and plans based on temporal logic. However, unlike [41, 59], in our centralized approach we minimize navigation distance rather than only satisfy the specifications. In [43], authors present decentralized control with synchronization signals for multi-robot systems. Such method must have strong synchronization during transition. However, in our decentralized synthesis framework, we only need robots send synchronization

signals when they have to.

Swarm control: To execute the synthesized plans, we can assign each individual robot with a decentralized plan and design continuous controllers to make robots achieve corresponding goals, as is shown in [55]. In addition, controlling swarm robots to execute the centralized plan can be considered as a task assignment problem, where robots are assigned to different goals and coordinated to complete the global task, e.g., multi-robot exploration in an unexpected environment [51], effective coordination and role assignment in formation control [84], and goal assignment for trajectory generation [61]. However, all these approaches only consider one specific goal and lacks the ability to perform high-level behaviors. In our work, we design a dynamic assigning mechanism during execution, which not only guarantees the correct high-level behaviors, but also returns an optimal solution.

Collision-free control: For the controller in continuous level, there are many approaches to multi-agent collision-free motion design (e.g. [11, 37, 39]), most of which consider collision avoidance as the primary goal. In this work, high-level symbolic plans are transformed into continuous controllers that implement them, as well as guaranteeing collision avoidance, bounded transition and deadlock mitigation. In [89], by integrating control barrier functions (CBF) and quadratic programming (QP), continuous controllers were designed to ensure collision avoidance. The proposed approach enables robots to move towards their objectives and execute collision avoidance controllers only when they have to, i.e., when some robots get too close to each other and might collide. In addition, robots need to avoid collisions with obstacles in the environment for safety. Similar ideas of barrier functions and calculating the safe and optimal control inputs are used in [89, 3, 85]. We extend control barrier functions to handle irregular obstacles, dynamic obstacles, and guarantee the correct transition at the symbolic level.

2.3 Preliminaries

In this section, we first introduce the syntax and semantics of linear temporal logic (LTL) [29], then we present the specifications and synthesis for swarm systems using LTL [55, 57], and finally we review control barrier function formulations [89] for swarm control that guarantee safety.

2.3.1 Linear Temporal Logic (LTL) and Specifications for Swarm Robots

Let AP be a set of Boolean propositions. We define the syntax of LTL as follows:

$$\varphi ::= \pi \,|\, \neg\varphi \,|\, \varphi \vee \varphi \,|\, \bigcirc \varphi \,|\, \varphi \mathscr{U} \varphi$$

where $\pi \in AP$ is a proposition, $\neg$ is negation, $\vee$ is disjunction, $\bigcirc$ is next and $\mathscr{U}$ is until. Other logical operators such as conjunction ($\wedge$), implication ($\Rightarrow$) and temporal operators such as *always* ($\Box$) and *eventually* ($\Diamond$) can be derived from these basic operators. An LTL formula over propositions AP is interpreted over infinite words $w \in (2^{AP})^{\omega}$. The language of an LTL formula φ, denoted by $\mathscr{L}(\varphi)$, is the set of infinite words that satisfy φ, i.e., $\mathscr{L}(\varphi) = \{w \in (2^{AP})^{\omega} \mid w \models \varphi\}$. Intuitively, $\Box\pi$ means proposition π has to be true all the time, $\Diamond\pi$ means proposition π will be true at some point, and $\bigcirc\pi$ means that proposition π has to be true in the next step. We refer the readers to [70] for more details about LTL.

In [55], we use a fragment of LTL formula, GR(1) [12], to write specifications for swarm robots at two levels: a *macroscopic* specification φ^M that describes how *groups* of robots should behave, and a *microscopic* specification φ^μ that describes how *individual*

robots must behave. We capture two kinds of properties allowed in the specifications: *safety* properties which states that "something bad never happens", and *liveness* requirements which indicate "something good eventually happens". More formally, assume $\mathbf{R} = \{r_1, \cdots, r_k\}$ is a set of regions partitioning the workspace. Let π_r be a proposition that is true *iff* at least one robot is currently in region $r \in \mathbf{R}$. Moreover, to distinguish between groups or *individual* robots, a parametric proposition is introduced for each region. Let $\pi_r^{\mathbf{a}}$ be a proposition that is true *iff* the robot $\mathbf{a}$ is in region $r \in \mathbf{R}$. The macroscopic specification φ^M is given as a temporal logic specification over region propositions $\Pi = \{\pi_{r_1}, \cdots, \pi_{r_k}\}$. The microscopic specification φ^μ is given as conjunction of formulas $\forall \mathbf{a}.\ \square\Diamond(\pi_{r_i}^{\mathbf{a}})$ and $\forall \mathbf{a}.\square(\pi_{r_j}^{\mathbf{a}})$, where $\forall \mathbf{a}.\ \square\Diamond(\pi_{r_i}^{\mathbf{a}})$ means that all robots must repeatedly visit region r_i but not necessarily at the same time, and $\forall \mathbf{a}.\square(\pi_{r_j}^{\mathbf{a}})$ means that all robots must always be in region r_j. Furthermore, in this work, we assume that the geometry and topology of the workspace are given following [55].

2.3.2 Synthesis of Swarm Behaviors

In the framework proposed in [55, 57], we allow the user to input a region graph that represents the connectivity of the workspace, and specifications that indicate the objectives of the system. We first obtain a centralized symbolic plan. Then, we partition the centralized symbolic plan into several decentralized plans and assign the decentralized plans to robots, where robots with the same symbolic plan consist of a sub-swarm. The synthesis process automatically determines the number of required sub-swarms and those sub-swarms may have variable size. Finally, a synchronization skeleton for each decentralized controller is constructed that indicates when each sub-swarm must synchronize with other sub-swarms to satisfy the correct collective behaviors.

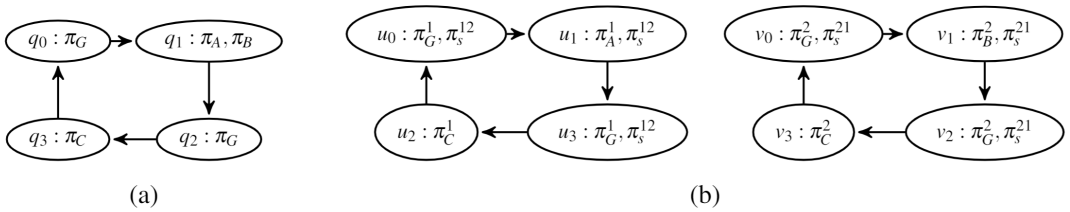


Figure 2.3: (a) A centralized symbolic plan. (b) Partitioning of the centralized plan of (a) into two decentralized plans. States q_i, u_j and v_k ($0 \leq i, j, k \leq 3$) are labeled with corresponding propositions that are true while other propositions false. For example, $u_0 : \pi_G^1, \pi_s^{12}$ represents $\pi_G \wedge \pi_s^{12} \wedge \neg\pi_A \wedge \neg\pi_B \wedge \neg\pi_C \wedge \neg\pi_D \wedge \neg\pi_E \wedge \neg\pi_F$. Propositions for synchronization can only appear in decentralized plans.

For example, given the workspace shown in Fig. 2.1, we can write a specification $\square\Diamond(\pi_A \wedge \pi_B \wedge \neg\pi_C \wedge \neg\pi_G) \wedge \square(\neg(\pi_C \wedge (\pi_A \vee \pi_B))) \wedge \forall \mathbf{a}.\square\Diamond\pi_C^{\mathbf{a}}$, where π_{r_i} represents at least one robot in region r_i ($r_i \in \{A, B, C, G\}$) and $\pi_C^{\mathbf{a}}$ indicates robot $\mathbf{a}$ in region C. The specification means:

- all robots should repeatedly only occupy regions A and B at the same time
- when there are robots in region C, no robot should be in region A or B, and vice versa
- every individual robot should visit region C at some point.

We specify the initial condition as $\pi_G \wedge \neg\pi_A \wedge \neg\pi_B \wedge \neg\pi_C \wedge \neg\pi_D \wedge \neg\pi_E \wedge \neg\pi_F$ which indicates that all robots initially locate in region G. Then, we can synthesize centralized and decentralized symbolic plans in Fig. 2.3(a) and Fig. 2.3(b) respectively. The synthesized plan in Fig. 2.3(a) indicates that the swarm robots, initially in G, move to A and B, and then move towards C through G. Such behavior is repeated indefinitely. The decentralized plans in Fig. 2.3(b) show the behaviors of two sub-swarms after partitioning. Two sub-swarms execute $G \to A \to G \to C$ and $G \to B \to G \to C$ respectively, where π_s^{kl} ($k, l \in \{1, 2\}, k \neq l$) indicates that sub-swarms k and l should synchronize at these states, and π_r^m ($m \in \{1, 2\}, r \in \{A, B, C, G\}$) indicates the propositions for sub-swarm m. Note

that the overall behavior of the decentralized plans is the same as the centralized plan although there is no communication between sub-swarms except the synchronization signals.

2.3.3 Control Barrier Functions for Continuous Control Synthesis

In this section, we summarize formulation of *control barrier functions* [89] to guarantee collision avoidance. We consider a swarm robotic system with N planar mobile robots, which are indexed by $\mathscr{M} = \{i \,|\, i = 1, 2, ..., N\}$. Every robot is modeled as a first-order system: $\dot{\mathbf{p}}_i = \mathbf{u}_i$, where $\mathbf{p}_i \in \mathbb{R}^2$ represents the position and $\mathbf{u}_i \in \mathbb{R}^2$ represents the velocity inputs of agent i. The velocity of agent i is bounded by $||\mathbf{u}_i||_\infty \leq \alpha_i$, where α_i is the maximum speed of agent i. Note that although we choose the first-order system as the model to fit kinematics of the physical robots in the demonstration, control barrier function can also be easily applied to higher-order systems [89]. To ensure collision avoidance, we require that any two robots keep a safety distance D_s away from each other at all times. That yields

$$||\Delta \mathbf{p}_{ij}|| \geq D_s \tag{2.1}$$

where $\Delta \mathbf{p}_{ij} = \mathbf{p}_i - \mathbf{p}_j$ as shown in Fig. 2.4(a).

To execute a symbolic plan, we synthesize control inputs at the continuous level to drive the robots to visit regions in sequence. We first represent the region by using its centroid point, and then map the symbolic plan to a sequence of centroid points. A simple continuous controller for every robot is a vector pointing to the target centroid point and we define the controller as the *nominal controller*.

Consider the dynamics of the entire swarm system as $\dot{\mathbf{x}} = \mathbf{u}$, where $\mathbf{x} = [\mathbf{p}_1^T, ..., \mathbf{p}_N^T]^T$ is the $2N$ dimensional position vector for all robots, and $\mathbf{u} = [\mathbf{u}_1^T, ..., \mathbf{u}_N^T]^T$ is the $2N$

dimensional control input (velocity vector). Following (2.1), we define a pairwise safe set ζ_{ij} for every two robots

$$\begin{aligned} &\zeta_{ij} = \{\mathbf{p} \in \mathbb{R}^2 \,|\, h_{ij}(\mathbf{p}) \geq 0\} \quad \forall i \neq j, \\ &h_{ij}(\mathbf{p}) = ||\Delta \mathbf{p}_{ij}|| - D_s. \end{aligned} \tag{2.2}$$

The safe set ζ for the swarm system is the intersection of all pair-wise safe set ζ_{ij}

$$\zeta = \prod_{i \in M} \{ \bigcap_{\substack{j \in M \\ j \neq i}} \zeta_{ij} \} \tag{2.3}$$

where the product is the Cartesian product of the state space of all robots, resulting in $2N$ dimensional ζ. When the state of the swarm is in ζ, the distance between any two robots is at least D_s.

To guarantee safety of the system at all times, the safe set needs to be *forward invariant*: if the system starts from the safe set, it always stays in the safe set, i.e., if $\mathbf{x}(0) \in \zeta$, then $\mathbf{x}(t) \in \zeta, \forall t \geq 0$. The *control barrier function* (CBF) is a function defined over state and input such that when it satisfies a set of constraints, it ensures the safe set is forward invariant [92]. Specifically, we define the pair-wise CBF as $B_{ij}(\mathbf{x}, \mathbf{u}) = \frac{dh_{ij}(\mathbf{x})}{dt} + \gamma h_{ij}^3(\mathbf{x})$, where γ is an arbitrary positive number. Note that $h_{ij}(\mathbf{x}) = 0$ is the boundary of the safe set. Combined with (2.2), the pair-wise CBF can be calculated as

$$B_{ij}(\mathbf{x}, \mathbf{u}) = \frac{\Delta \mathbf{p}_{ij}^T}{||\Delta \mathbf{p}_{ij}||} \Delta \mathbf{u}_{ij} + \gamma h_{ij}^3 \tag{2.4}$$

where $\Delta \mathbf{u}_{ij} = \mathbf{u}_i - \mathbf{u}_j$. As long as the control input $\mathbf{u}$ satisfies $B_{ij}(\mathbf{x}, \mathbf{u}) \geq 0$, $\forall i, j \; i \neq j$, the forward invariance of safe set ζ can be guaranteed. We refer readers to [89] for more details and proofs.

We consider all the robots simultaneously to synthesize control inputs such that all pair-wise inequality constraints in (2.1) are satisfied. Note that there are $\frac{N(N-1)}{2}$

inequalities $B_{ij}(\mathbf{x},\mathbf{u}) \geq 0$, leading to a set of linear constraints $A_{ij}\mathbf{u} \leq b_{ij}$, where $A_{ij} = [0,...,-\Delta\mathbf{p}_{ij}^T,...,\Delta\mathbf{p}_{ij}^T,...,0]$, $\mathbf{u} = [\mathbf{u}_1^T,\mathbf{u}_2^T,...,\mathbf{u}_N^T]^T$ is the joint control inputs for all robots, and $b_{ij} = \gamma h_{ij}^3 ||\Delta\mathbf{p}_{ij}||$. Given such constraints, we use quadratic programming (QP) to modify nominal controllers as little as possible to ensure safety (collision avoidance) and liveness (reaching the primary goal). We formulate the QP problem for centralized control [89] as

$$\begin{aligned} \mathbf{u}^* = \underset{\mathbf{u}\in\mathbb{R}^{2N}}{\operatorname{argmin}} \quad & J(\mathbf{u}) = \sum_{i=1}^{N} ||\mathbf{u}_i - \hat{\mathbf{u}}_i||^2 \\ s.t. \quad & A_{ij}\mathbf{u} \leq b_{ij}, \quad \forall i \neq j \\ & ||\mathbf{u}_i||_\infty \leq \alpha_i, \quad \forall i \in \mathscr{M}, \end{aligned} \tag{2.5}$$

where $\mathbf{u}_i$ is the actual control input and $\hat{\mathbf{u}}_i$ is the nominal control input for agent i, respectively. By minimizing the difference between the nominal and actual control inputs, we can synthesize continuous control inputs to guarantee collision avoidance.

To decentralize the computation, we can find sufficient conditions for constraints in (2.5) to obtain (2.6) and (2.7):

$$-\Delta\mathbf{p}_{ij}^T\mathbf{u}_i \leq \frac{\alpha_i}{\alpha_i+\alpha_j} b_{ij} \tag{2.6}$$

$$\Delta\mathbf{p}_{ij}^T\mathbf{u}_j \leq \frac{\alpha_j}{\alpha_i+\alpha_j} b_{ij} \tag{2.7}$$

where i and j represent robot i and robot j, and α_i and α_j are control input limits of robot i and robot j, respectively. Note that switching i and j in (2.6) gives the same constraint as (2.7). Such inequalities distribute the collision avoidance control to each robot according to their agility. Each robot can run its own QP solver:

$$\begin{aligned} \mathbf{u}_i^* = \underset{\mathbf{u}_i\in\mathbb{R}^2}{\operatorname{argmin}} \quad & J(\mathbf{u}_i) = ||\mathbf{u}_i - \hat{\mathbf{u}}_i||^2 \\ s.t. \quad & A'_{ij}\mathbf{u}_i \leq b'_{ij}, \quad \forall j \neq i \\ & ||\mathbf{u}_i||_\infty \leq \alpha_i \end{aligned} \tag{2.8}$$

where $A'_{ij} = -\Delta\mathbf{p}_{ij}^T\mathbf{u}_i$ and $b'_{ij} = \frac{\alpha_i}{\alpha_i+\alpha_j}b_{ij}$. In this case, instead of solving a $2N$-dimensional QP, 2-dimensional QPs in terms of $\mathbf{u}_i$ are solved for each robot i. Note that a fixed sensing range is applied for each robot such that the robot only uses the sensed positions of others robots, and the collision avoidance can still be guaranteed. Detailed formulations for such QP can be found in [89].

2.4 Continuous Control Synthesis

When executing the symbolic plans on physical robots, the continuous controllers are required to guarantee the following properties: (i) *correct behavior*: the robot trajectories are correct mapping from the synthesized symbolic plans to the continuous space, (ii) *collision avoidance*: robots must avoid collision with each other and obstacles in their environment, (iii) *region invariance*: if robots are moving from region r_i to region r_j, they stay within the boundaries of regions r_i and r_j and do not enter any other region $r \in \mathbf{R}\backslash\{r_i, r_j\}$ until the transition is complete, thereby continuously implementing the discrete abstraction regarding the motion, (iv) *deadlock mitigation*: robots should be able to escape most deadlock situations. In this section, we extend CBFs and synthesize continuous controls that guarantee collision free motions with polygonal or dynamic obstacles in the environment. In addition, we briefly discuss a method to maintain region invariance and present a roadmap-based approach to mitigating one-robot-one-wall deadlocks.

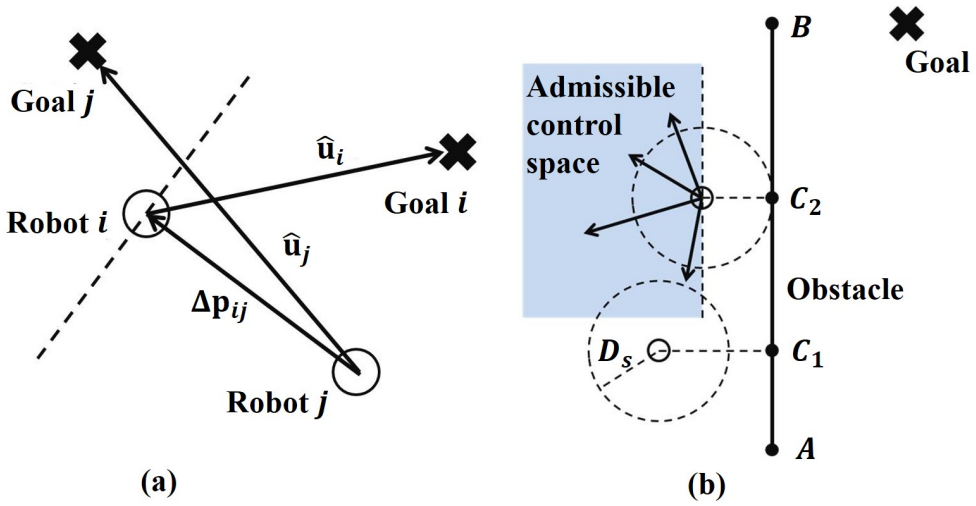


Figure 2.4: (a) Relative positions between two robots. (b) Relative positions between the robot and the line obstacle.

2.4.1 Static/dynamic circular obstacle avoidance

To ensure safety of the system, swarm robots must not only avoid collision with each other, but also with static and dynamic obstacles in their environment. In [89], the authors mentioned that circular obstacles can be avoided by regarding obstacles as virtual robots with 0 control inputs. Here, for first-order systems, we extend the work to collision avoidance with dynamic obstacles which are modeled as virtual robots with bounded control inputs.

Let α_i and β_j be the velocity bound of robot i and dynamic obstacle j respectively $(\alpha_i, \beta_j > 0)$. We assume that $\forall i, j, \beta_j \leq \alpha_i$, otherwise the obstacle can always move toward robot i and cause a collision. Then, we revise the constraint in section 2.3.3 to be

$$-\Delta\mathbf{p}_{ij}^T\mathbf{u}_i \leq b_{ij} - ||\Delta\mathbf{p}_{ij}||\beta_j \tag{2.9}$$

where we regard obstacle j as a virtual robot with position $\mathbf{p}_j$. We add all linear constraints (2.9) to the decentralized QP to synthesize collision-free continuous controllers

for each robot.

Note that when considering two robots, both controllers actively avoid collisions; however in the case of a dynamic obstacle and a robot, the obstacle could have an arbitrary velocity within the bound β_j. Equation (2.9) provides the most conservative constraint for robot i and guarantees a non-negative CBF $\mathbf{B}_{ij}$ for robot i and obstacle j. In addition, $b_{ij} - ||\Delta\mathbf{p}_{ij}||\beta_j \geq -||\Delta\mathbf{p}_{ij}||\beta_j \geq -||\Delta\mathbf{p}_{ij}||\alpha_i$, which ensures that there are always feasible solutions for (2.9). In the case of static obstacles, we apply $\beta_j = 0$, which models the static obstacle as a robot with zero control input as mentioned in [89].

2.4.2 Static polygonal obstacle avoidance and Region Invariance

Section 2.4.1 describes collision avoidance with circular obstacles. However, in most cases, static obstacles such as walls cannot be approximated well by circles. Instead, polygons are better suited to represent irregular obstacles. Hence, in realistic environments, we need to leverage the CBFs to make robots avoid collisions with polygons. In this section we define safe sets and develop the appropriate barrier functions to ensure no collisions with polygonal obstacles.

Definition 3.1: For a point obstacle C in the workspace, the safe set ζ_C of point C is the set of robot positions such that the distance between the robot and point C is at least the safety distance D_s, i.e., $\zeta_C = \{\mathbf{p} \in \mathbb{R}^2 \,|\, ||\mathbf{p} - \mathbf{p}_C|| \geq D_s\}$, where $\mathbf{p}$ is the position of the robot, $\mathbf{p}_C$ is the position of point C and D_s is the safety distance.

Definition 3.2: The safe set of a straight line AB is the union of the safe sets of all points on that line, i.e., $\zeta_{AB} = \cup_{C'=A}^{B} \zeta_{C'}$, where C' is a point on the line segment AB.

Given a line segment ζ_{AB} and the closest point C on ζ_{AB}, the robot is guaranteed to

be in the safe set of ζ_{AB} if the controller ensures that the robot is in the safe set ζ_C of C.

Proof: Consider a single robot, a line obstacle and a goal position for the robot, as is shown in Fig. 2.4 (b). We draw a circle centered at the robot position with a radius of the safety distance D_s. If the robot is in the safe set of the closest point, then the circle has no intersection with the line, which means the robot is currently in the safe set of the line. For the robot pose after applying the control, we consider the most "dangerous" case where the robot is distance D_s away from the line, which indicates that the circle is tangential to the line. As the synthesized controller ensures that the robot is still in the safety set of the closest point, the actual control input can only be in the blue (shaded) region. Therefore, the robot cannot get any closer to the line no matter where the goal position is, thereby ensuring that the robot is always in the safe set of the entire line.

Based on Proposition 2.4.2, we add constraints by dividing all the polygonal obstacles into line segments, finding the closest point to each line segment, and considering those points as virtual robots with no control inputs. Then, we formulate CBFs in terms of virtual robots as well as real robots and add those constraints to the original QP (2.5) to compute the control inputs for collision-free motion.

To guarantee the region invariance, when a sub-swarm is moving between two regions, we construct virtual obstacles on the boundaries of the union of these two regions and follow the procedure above to modify the QP with additional constraints. In the example shown in Fig. 2.1, when sub-swarms are required to move from region G to region A, we construct a virtual obstacle at the entrances of all regions other than G and A, and use the CBF with line segments to ensure the robots do not enter B or any other region.

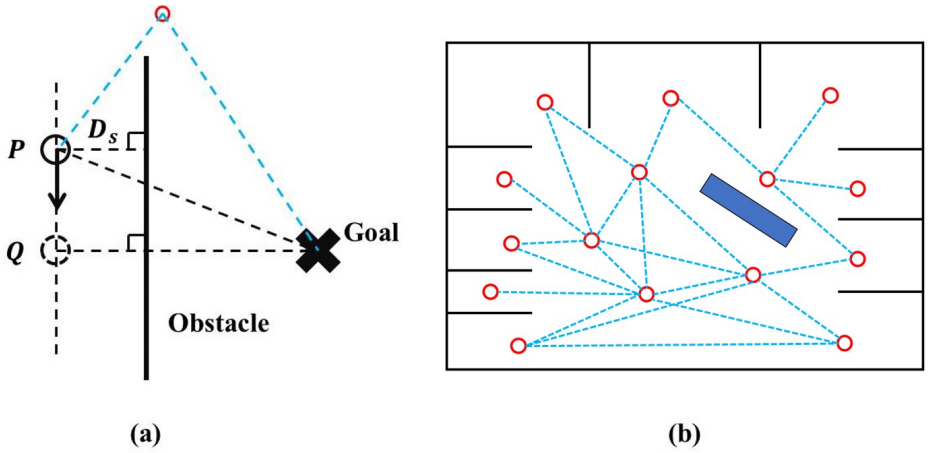
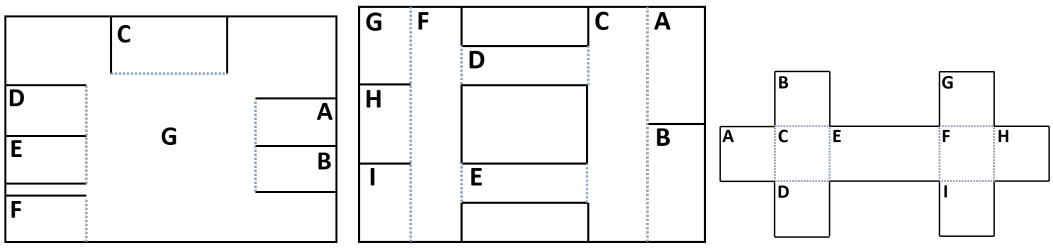


Figure 2.5: (a) Deadlock scenario when there is a line obstacle between the robot and its goal. (b) Road map over the work space, where red circles represent nodes and blue dashed lines represent edges.

2.4.3 Deadlock mitigation

Deadlock is defined as a situation where no progress can be made towards the goal. Specifically, swarm robots might get into a deadlock situation when the effects of the barrier functions counteract the nominal controller. Consistent perturbations are used in [89] to solve the deadlock problem between the robots themselves, where signals that are tangential to the nominal control are given to drive the robots around. However, the same strategy might not work when dealing with polygonal obstacles. Figure 2.5(a) shows a possible deadlock scenario: when the robot is at point P, the control input drives the robot "closer" to the goal (towards the point Q). However, when the robot reaches Q, it cannot make progress toward the goal position any more. Note that giving a perturbation does not help the robot get out of the deadlock situation as the robot always tends to move to Q if the wall is between the robot and the goal.

In a single-robot-single-wall deadlock scenario, the robot is exactly D_s away from the wall and the line connecting the robot to the goal perpendicularly intersects the wall

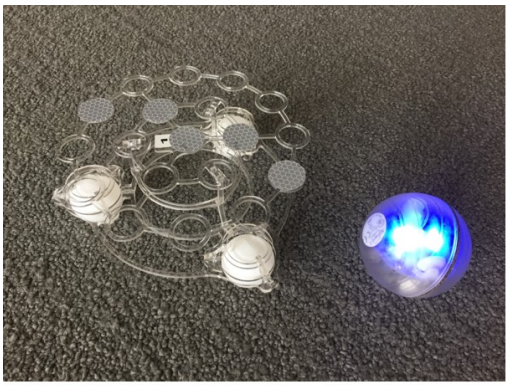


(a) Workspace for Example 2.4.1 (b) Workspace for Example 2.4.2 (c) Workspace for Example 2.6.3

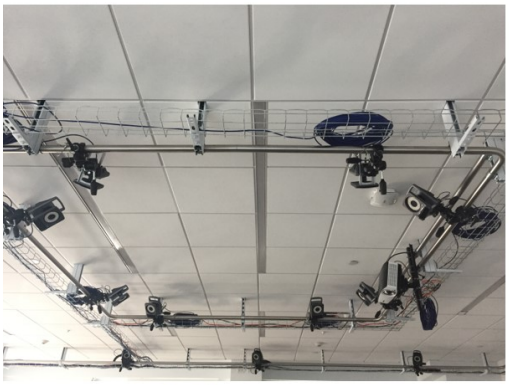
Figure 2.6: Workspace for demonstrations with physical platforms.

(e.g. a robot at point Q in Fig. 2.5 (a) is in deadlock). We can add a waypoint to make the robot go around the wall to reach the goal. Note that the robot will never go into a deadlock if the line connecting the robot and the goal has no intersection with the wall. Thus, we revise the nominal controller (replan the path) whenever we detect a wall between the robot and its goal. To this end, with a priori knowledge of the workspace, we build a roadmap [5] over the workspace as shown in Fig. 2.5(b), and use any roadmap-based motion planning method [86] to revise temporal goals to guarantee a deadlock-free path. Note that when applying this deadlock mitigation technique, we ensure that the roadmap does not create paths that violate region invariance as we add virtual walls to the environment before we search the path in the roadmap. Robots will never get stuck in a deadlock caused by static polygonal obstacles with the roadmap.

In the multi-robot-multi-wall case, deadlocks can be mitigated by tangential perturbation [89] together with the roadmap-based planning, although we cannot guarantee deadlock avoidance in all situations.



(a) Sphero (right) and rolling cage (left)



(b) Vicon system

(c) Robotarium

Figure 2.7: Experimental setup for demonstrations with physical platforms in Examples 2.4.1, 2.4.2, and 2.6.3.

2.4.4 Physical Demonstration with Spheros

Physical Setup: SPRK robots[1] are rolling sphere robots with a diameter of 7.3 cm, shown in Fig. 2.7(a). The robots communicate through Bluetooth and are programmed using the Robot Operating System (ROS) [74]. The control inputs are linear velocities in two perpendicular directions. Thus, the SPRK robots can be modeled as holonomic first-order systems. We use the Vicon motion capture system to localize the robots, and design rolling cages to enable placing markers on the robots as shown in Fig. 2.7(a).

Example 2.4.1. This example shows the ability of the swarm to execute decentralized

[1] https://www.sphero.com/sphero

symbolic plans synthesized from high-level specifications. In this example, we focus on the correct and safe behaviors at the continuous level with a dynamic obstacle randomly moving in the workspace. We present the demonstration with 11 SPRK robots.

Consider a workspace divided into 7 regions ($A-G$) as shown in Fig. 2.6a. All robots are initially positioned on the right side of region G. robots are required to: repeatedly visit region C (could be at different times) ($\varphi_1^{\mu} = \forall \mathbf{a}.\, \square\Diamond \pi_C^{\mathbf{a}}$), occupy regions D, E, and F at the same time ($\phi_{11} = \square\Diamond(\pi_D \wedge \pi_E \wedge \pi_F \wedge \bigwedge_{r\in R\setminus\{D,E,F\}} \neg\pi_r)$), occupy regions A and B at the same time ($\phi_{12} = \square\Diamond(\pi_A \wedge \pi_B \wedge \bigwedge_{r\in R\setminus\{A,B\}} \neg\pi_r)$), and if any robots are in A or B, there should not be any robots in D, E, or F and vice versa ($\phi_{13} = \square(\neg((\pi_A \vee \pi_B) \wedge (\pi_D \vee \pi_E \vee \pi_F)))$). The macroscopic specification is defined as $\varphi_1^M = \phi_{11} \wedge \phi_{12} \wedge \phi_{13}$. We synthesize three decentralized plans $\mathscr{T}_{11}, \mathscr{T}_{12}, \mathscr{T}_{13}$ with the following runs and implement the plans on three groups of robots:

- $\mathscr{T}_{11}$: $(\dot{\pi}_G \to \pi_C \to \pi_G \to \dot{\pi}_D \to \pi_G \to \dot{\pi}_A)^{\omega}$,
- $\mathscr{T}_{12}$: $(\dot{\pi}_G \to \pi_C \to \pi_G \to \dot{\pi}_E \to \pi_G \to \dot{\pi}_A)^{\omega}$,
- $\mathscr{T}_{13}$: $(\dot{\pi}_G \to \pi_C \to \pi_G \to \dot{\pi}_F \to \pi_G \to \dot{\pi}_B)^{\omega}$.

The symbolic plans loop back to the initial state after visiting their last state and the three groups synchronize at states denoted with a dot.

Results: Figure 2.8 demonstrates the implementation of Example 2.4.1. The SPRK robots were divided into three groups, each assigned a decentralized plan and a color to display (blue is $\mathscr{T}_{11}$, green is $\mathscr{T}_{12}$, and red is $\mathscr{T}_{13}$). Each group executed a decentralized automaton and a synchronization skeleton in the continuous 2D space that guaranteed the correct global behaviors. In addition, we manually controlled a separate SPRK robot (circled yellow robot) to model a dynamic obstacle. The results show that all robots guarantee the collision-free motion, region invariance as well as the deadlock

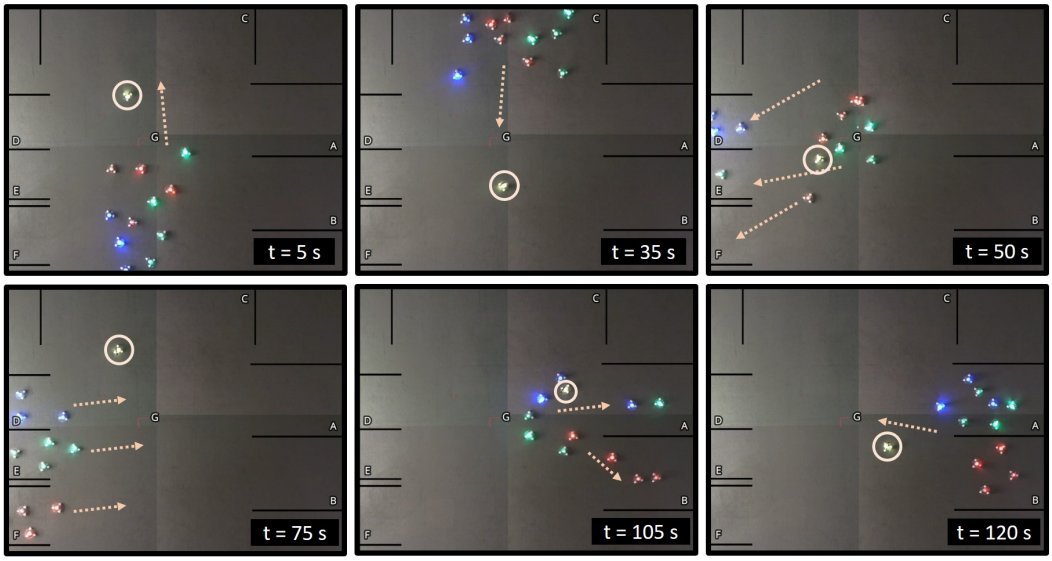


Figure 2.8: Example 2.4.1. SPRKs are divided into three groups specified by green, blue and red. Lines represent walls in the workspace and dashed arrows indicate the motion. The dynamic obstacle is the circled yellow robot.

mitigation, with a dynamic obstacle moving in the workspace. More details are in the submitted video.

2.5 Centralized and Decentralized CBF with Individual Failures

In this section, we compare the centralized and decentralized CBF in the presence of individual robot failures. We present different behaviors in Robotarium [65] using the centralized and decentralized versions of CBF for the continuous control.

2.5.1 Centralized and Decentralized CBF during Individual Failures

In section 2.3, we describe the centralized and decentralized versions of CBF which guarantee collision avoidance. However, when individual robots fail in a swarm, constraints in the QP might be violated. For example, the centralized CBF requires (2.4) to be non-negative, which yields:

$$\Delta \mathbf{p}_{ji}^T \mathbf{u}_i + \Delta \mathbf{p}_{ij}^T \mathbf{u}_j \leq b_{ij}. \tag{2.10}$$

In the case that $\Delta \mathbf{p}_{ji}^T \mathbf{u}_i$ is positive and $\Delta \mathbf{p}_{ij}^T \mathbf{u}_j$ is negative, the failure of robot j increases the value of the left side of (2.10), and the constraint (2.10) may be violated.

In the decentralized CBF, individual failures do not lead to collision. Note that $b_{ij} = \gamma h_{ij}^3 ||\Delta \mathbf{p}_{ij}||$ is always positive. We first assume that robot j fails to execute control input and stays static, which indicates that $\mathbf{u}'_j = \mathbf{0}$ and $\alpha'_j = 0$, where $\mathbf{u}'_j$ and α'_j are the actual control inputs and limits. It is obvious that inequality (2.7) is satisfied as both sides are 0. Furthermore, as

$$-\Delta \mathbf{p}_{ij}^T \mathbf{u}_i \leq \frac{\alpha_i}{\alpha_i + \alpha_j} b_{ij} \leq b_{ij} = \frac{\alpha_i}{\alpha_i + \alpha'_j} b_{ij},$$

inequality (2.6) is also satisfied for the actual system. Additionally, it is obvious that assuming a failure in robot i instead of robot j, or failures in both robot i and j, also leads to satisfaction of both (2.6) and (2.7). Thus, the decentralized computation at the continuous level is more tolerant (i.e., still guarantees collision avoidance with some robots failing to move) to possible failures during actual execution of controllers.

Figure 2.9 illustrates the scenario more intuitively, where robot i is "chasing" robot j. If robot j encountered failure and stopped, robot i might collide with robot j as the centralized CBF has a strong assumption that robot j must apply a velocity $\mathbf{u}_j$ to avoid

collision. However, decentralized CBF guarantees the collision-free motion even though robot j stops to move.

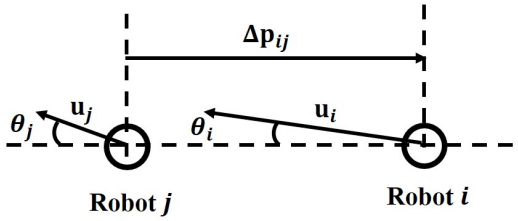


Figure 2.9: A scenario where a transient failure of robot j may violate the centralized CBF and cause collision but will not violate the decentralized CBF.

2.5.2 Demonstrations of Centralized and Decentralized CBFs in Robotarium

Physical setup: Robotarium is a remotely accessible swarm testbed [65] where GRITSBots serve as swarm robots shown in Fig. 2.7c. GRITSBots are 3×3.1 cm wheeled robots which can be modeled as unicycles. They are controlled through wireless communication and are localized by web cameras [66].

Example 2.5.1. The example considers two working zones connected by two long corridors as shown in Fig. 2.6b. The swarm, initially distributed in region C, is required to infinitely often navigate through corridors and visit regions G, H, and I simultaneously $(\phi_{21} = \square\lozenge(\pi_G \wedge \pi_H \wedge \pi_I \wedge \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{H,G,I\}} \neg\pi_{\mathbf{r}}))$. The whole swarm must also repeatedly be in region F $(\phi_{22} = \square\lozenge(\pi_F \wedge \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{F\}} \neg\pi_{\mathbf{r}}))$, and infinitely often occupy regions A and B at the same time $(\phi_{23} = \square\lozenge(\pi_A \wedge \pi_B \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{A,B\}} \neg\pi_{\mathbf{r}}))$. Moreover, all the robots in the swarm must pass through corridor E repeatedly $(\varphi_2^{\mu} = \forall\mathbf{a}.\ \square\lozenge(\pi_E^{\mathbf{a}}))$ but they must avoid occupying both corridors at any time $(\phi_{24} = \square(\neg(\pi_E \wedge \pi_D)))$. Finally, if there is a robot in any regions $\{G,H,I\}$, there must be no robots in regions $\{A,B\}$ and vice versa

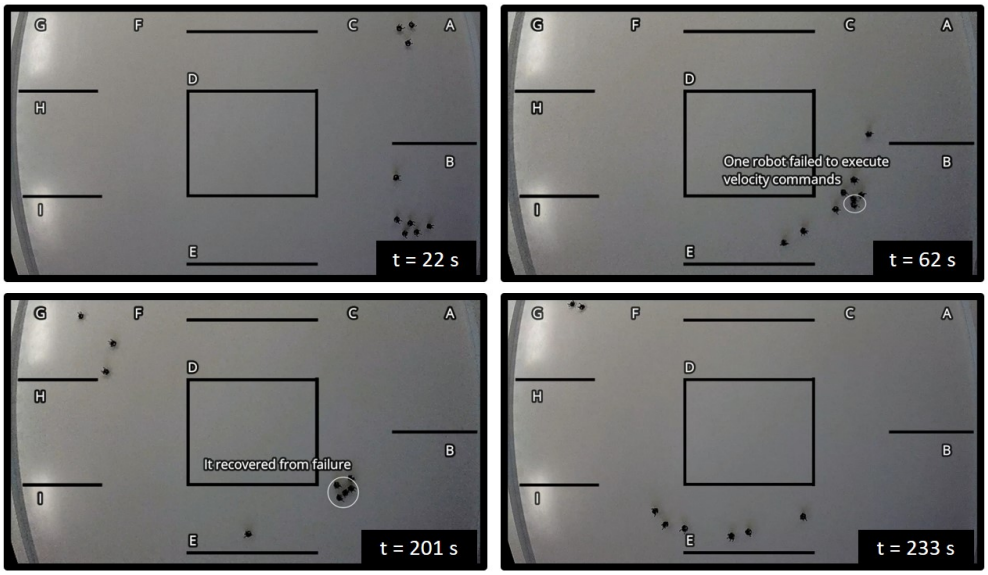


Figure 2.10: Example 2.5.1. Robots are divided into three groups. Black lines represent walls in the workspace. The continuous control uses centralized CBF. Failure happened on $t = 62s$ and collision happened before $t = 201s$. Robots successfully completed the task after the failed robot recovered.

$(\phi_{25} = \square(\neg((\pi_G \vee \pi_H \vee \pi_I) \wedge (\pi_A \vee \pi_B))))$. The macroscopic specification is defined as $\varphi_2^M = \bigwedge_{i=1}^{5} \phi_{2i}$. We synthesize three decentralized symbolic plans $\mathscr{T}_{21}, \mathscr{T}_{22}, \mathscr{T}_{23}$ with the following runs and implement the plans on three groups of robots:

- $\mathscr{T}_{21}$: $(\dot{\pi}_A \to \pi_C \to \pi_E \to \pi_F \to \dot{\pi}_I \to \dot{\pi}_F \to \pi_E \to \pi_C)^\omega$,
- $\mathscr{T}_{22}$: $(\dot{\pi}_B \to \pi_C \to \pi_E \to \pi_F \to \dot{\pi}_G \to \dot{\pi}_F \to \pi_E \to \pi_C)^\omega$,
- $\mathscr{T}_{23}$: $(\dot{\pi}_B \to \pi_C \to \pi_E \to \pi_F \to \dot{\pi}_H \to \dot{\pi}_F \to \pi_E \to \pi_C)^\omega$.

As before, the symbolic plans loop back to their initial state after visiting their last state and they synchronize with each other in the regions marked with a dot.

Results: We implement this example in the Robotarium platform [65] with both centralized and decentralized CBFs. During task execution, one robot failed to execute the given velocity command and stayed still. The failed robot blocked the motion of

other robots and caused collision using centralized CBF while there was never a collision using decentralized CBF. Figure 2.10 shows the implementation of Example 2.5.1 on the Robotarium platform, where centralized and decentralized CBF are used in two trials separately. Nine robots start from region C, first occupy regions A and B, then move through region E to occupy regions G, H, and I, and move back to C. During the demonstration with the centralized CBF, one robot failed to move due to network delays or wheel slips at $t = 62s$. Such individual failure caused collisions of five robots as the centralized barrier functions have been violated. After the robot recovered from the failure and was able to execute the correct velocity command, the whole swarm made progress again and completed the high-level task eventually. More details are in the submitted video.

2.6 Dynamic Assignment for Centralized Symbolic Plans

The synthesized decentralized symbolic plans [55, 57] guarantee the correct behaviors with respect to the high-level specifications. However, we can further improve the efficiency of achieving high-level tasks by using the geometry information of the workspace and rearranging the destination of each robot. In this section, we describe the centralized execution of symbolic plans in a correct and efficient way by dynamically assigning robots to different parts of the symbolic plan. In addition, we compare the execution of centralized and decentralized symbolic plans using MATLAB simulation and Robotarium platform [65], and discuss the execution efficiency, failure resilience and computation complexity.

2.6.1 Dynamic Goal Assignment

In the approach of synthesizing decentralized symbolic plans in section 2.4, we divide all robots into sub-swarms of fixed sizes before execution with each robot executing its own plan. The swarm only needs global coordination when there is a synchronization signal. However, since we typically consider each robot as interchangeable in swarm systems, robots could exchange their plans during transitions to conduct the task more efficiently while achieving the same high-level goals. For example, in example 2.4.1, at $t = 50s$ in Fig. 2.8, robots in the red sub-swarm and green sub-swarm could have exchanged their plans instead of crossing each other's paths slowly due to collision avoidance. In addition, if some robots fail in a sub-swarm during execution of the decentralized symbolic plans, the entire swarm might no longer complete the task as the remaining number of robots could violate the minimal size of the sub-swarm. For example, if a sub-swarm with only one robot fails, the entire swarm cannot make progress to the goal anymore. In contrast, with centralized execution of the symbolic plan, the other robots can automatically fill in the roles for the failing robots, since the plans are assigned dynamically during execution. Motivated by efficiency and failure resilience, we propose a dynamically assigning method to directly execute the centralized symbolic plan using integer programming (IP).

We consider transitions in centralized symbolic plans and regard the two states in each transition as the current state and the next state respectively. Each state is labeled with a set of Boolean variables indicating whether there are robots in the corresponding region or not. For example, given the workspace in Fig. 2.1 with regions $A - G$, the state vector $[1,1,0,0,0,0,0]^T$ represents that robots only occupy regions A and B. We formalize the centralized assigning problem as follows.

We assume that there are N robots executing a centralized symbolic plan in n regions.

First, we define an assigning matrix $M \in \mathbb{R}^{N\times n}$ as

$$[M]_{ij} = \begin{cases} 1 \text{ (if robot } i \text{ is assigned to} \\ \text{region } j \text{ in the next state)} \\ 0 \text{ (otherwise).} \end{cases} \tag{2.11}$$

The goal is to find an assigning matrix M such that the resulting assignment can guarantee a correct and efficient transition. Let $\mathbf{p}_i$ represent the current position of robot i and $\mathbf{p}_{ij}^r$ represent the closest point in region j to the current position of robot i ($1 \leq i \leq N$, $1 \leq j \leq n$). We define $C_{ij} = ||\mathbf{p}_i - \mathbf{p}_{ij}^r||$ as the shortest distance for robot i moving to region j. We formulate the problem as an integer program (IP):

$$M = \underset{[M]_{ij}\in\{0,1\}}{\operatorname{argmin}} \quad \sum_{i=1}^{N}\sum_{j=1}^{n} M_{ij}C_{ij} \tag{2.12a}$$

$$s.t. \quad \sum_{j=1}^{n} M_{ij} = 1 \quad \forall 1 \leq i \leq N \tag{2.12b}$$

$$\sum_{i=1}^{N} M_{ij} \geq N_j^{min} \quad \forall 1 \leq j \leq n \tag{2.12c}$$

$$\sum_{i=1}^{N} M_{ij} \leq N_j^{max} \quad \forall 1 \leq j \leq n \tag{2.12d}$$

$$M_{ij} = 0 \quad \text{(if the current region of} \\ \text{robot } i \text{ is not connected to region } j\text{)}. \tag{2.12e}$$

The IP formulation in (2.12) finds the assigning matrix M to minimize the cost while guaranteeing the correct behaviour. The meaning of each equation is shown as follows:

- (2.12a): The cost for a transition is the sum of distances that all robots must travel for entering the target regions.
- (2.12b): each robot must be assigned to only one region in the next state.
- (2.12c): the number of robots assigned to region j must be no-less than N_j^{min},

where N_j^{min} is the user-specified minimum number of robots that can be in region j.

- (2.12d): the number of robots assigned to region j must be no-greater than N_j^{max}, where N_j^{max} is the user-specified maximum number of robots that can be in region j.
- (2.12e): robots cannot be assigned to a non-neighboring region.

Note that $N_j^{min} \geq 1$ if π_j is true in the next state and that $N_j^{max} = 0$ if π_j is false in the next state, where π_j is the proposition that indicates whether region j is occupied by any robot(s). In addition, the minimum number of swarm size N_j^{min} can be automatically obtained according to [57]. The way to determine whether M_{ij} is 0 is to find the region i' where robot i is currently located, and then determine whether region i' and region j are connected in the region graph.

By satisfying all the constraints in (2.12b-2.12e), the transition is guaranteed to be completed correctly when there is no deadlock, and every robot will only be assigned to either the current region or the neighboring region. In addition, as the solution gives the robot assignment with the minimum travel distance, the robots can complete the transition in an efficient way. Note that we use the Euclidean distance to represent the cost function but the actual cost might differ from that since there can be obstacles in the environment. However, such assigning method can still return a correct and near-optimal solution which decreases the total traveling distance.

2.6.2 Comparison of Execution of centralized and decentralized symbolic plans in Simulation

We present two scenarios that compare the execution of the centralized and decentralized symbolic plans. Specifically, we demonstrate the differences in execution efficiency and computation complexity in the first scenario, and we present the difference in performances in the presence of failures in the second scenario. In the simulation, all robots are simulated as first-order systems. Note that we use the controller synthesis for collision avoidance, region invariance and deadlock mitigation mentioned in section 2.4 in the MATLAB simulation to demonstrate that both centralized and decentralized symbolic plans can be correctly executed.

Example 2.6.1. In this scenario, we compare the execution of the centralized and decentralized symbolic plans in terms of efficiency and computation complexity. In particular, we compare the distances that swarm robots move, the time they spend on completing the task, and the computation time by using the same task specifications and initial positions.

We consider a workspace divided into 11 regions (A - K) which are shown in Fig. 2.11. There are 15 robots initially located in region A. The task is to make all robots repeatedly visit regions B, F, I, repeatedly visit G, J, C, and repeatedly visit K, D, H at the same time. In addition, all robots repeatedly visit region A. We encode such task

descriptions in LTL:

$$\phi_{31} = \square\diamondsuit(\pi_B \wedge \pi_F \wedge \pi_I \wedge \bigwedge_{r\in R\setminus\{B,F,I\}} \neg\pi_r)$$

$$\phi_{32} = \square\diamondsuit(\pi_G \wedge \pi_J \wedge \pi_C \wedge \bigwedge_{r\in R\setminus\{G,J,C\}} \neg\pi_r)$$

$$\phi_{33} = \square\diamondsuit(\pi_K \wedge \pi_D \wedge \pi_H \wedge \bigwedge_{r\in R\setminus\{K,D,H\}} \neg\pi_r)$$

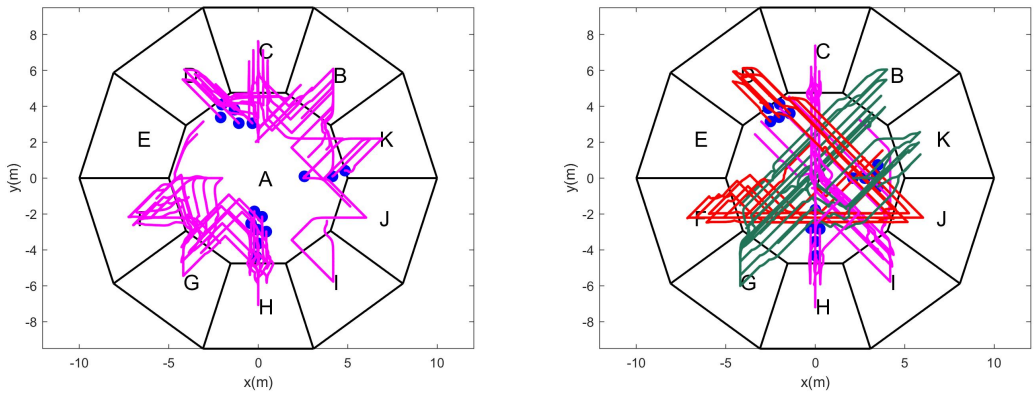
$$\phi_{34} = \square\diamondsuit(\pi_A \wedge \bigwedge_{r\in R\setminus\{A\}} \neg\pi_r)$$

The task specification is defined as $\phi_3 = \phi_{31} \wedge \phi_{32} \wedge \phi_{33} \wedge \phi_{34}$. Using the synthesis method mentioned in section 2.3, we first obtain a centralized symbolic plan $\mathscr{T}_3$: $(\pi_A \rightarrow \pi_B \wedge \pi_F \wedge \pi_I \rightarrow \pi_A \rightarrow \pi_G \wedge \pi_J \wedge \pi_C \rightarrow \pi_A \rightarrow \pi_K \wedge \pi_D \wedge \pi_H)^\omega$, where ω means that $\mathscr{T}_3$ loops back to the first state after reaching the last. In addition, we partition $\mathscr{T}_3$ and obtain 3 decentralized symbolic plans:

- $\mathscr{T}_{31}$: $(\dot{\pi}_A \rightarrow \dot{\pi}_B \rightarrow \pi_A \rightarrow \dot{\pi}_G \rightarrow \pi_A \rightarrow \dot{\pi}_K)^\omega$,
- $\mathscr{T}_{32}$: $(\dot{\pi}_A \rightarrow \dot{\pi}_F \rightarrow \pi_A \rightarrow \dot{\pi}_J \rightarrow \pi_A \rightarrow \dot{\pi}_D)^\omega$,
- $\mathscr{T}_{33}$: $(\dot{\pi}_A \rightarrow \dot{\pi}_I \rightarrow \pi_A \rightarrow \dot{\pi}_C \rightarrow \pi_A \rightarrow \dot{\pi}_H)^\omega$.

Each symbolic plan loops back to the initial state after visiting the last state, and the three groups synchronize at states denoted with a dot.

We execute the centralized plan $\mathscr{T}_3$ using the dynamic assigning method in section 2.6.1. In addition, we execute the decentralized plans by assigning the programs to robots offline (5 robots for each decentralized plan). The continuous controllers are synthesized using the method in section 2.4 to guarantee safe and correct execution. The resulting trajectories of the execution of the centralized and decentralized symbolic plans are both shown in Fig. 2.11. We use 10 different sets of initial positions and



(a) Centralized execution

(b) Decentralized execution

Figure 2.11: Trajectories of 15 simulated robots given the same high-level specifications, workspace, and set of initial conditions in example 2.6.1 by executing the centralized plan with dynamic assignment (a) and executing decentralized plans (b). Different colors in (b) represents trajectories of different sub-swarms.

compare the average traveling distances, execution time, and computation time in table 2.1.

Example 2.6.2. In this example, we show the different performances in both centralized and decentralized execution when unexpected failure happens during execution. We consider a workspace divided into 8 regions (A - H) shown in Fig. 2.12a and a team of 8 robots. All robots start from region G. They must repeatedly visit F (could be at different times), repeatedly visit C, D, and E at the same time, and repeatedly visit A and H at the same time. Additionally, there is a safety requirement that no robots can be in region B and G at the same time. We define the task in LTL as:

$$\varphi_4^{\mu} = \forall \mathbf{a}.\ \square\Diamond \pi_F^{\mathbf{a}}$$

$$\phi_{41} = \square\Diamond(\pi_C \wedge \pi_D \wedge \pi_E \wedge \bigwedge_{r\in R\setminus\{C,D,E\}} \neg\pi_r)$$

$$\phi_{42} = \square\Diamond(\pi_A \wedge \pi_H \wedge \bigwedge_{r\in R\setminus\{A,H\}} \neg\pi_r)$$

$$\phi_{43} = \square(\neg(\pi_B \wedge \pi_G))$$

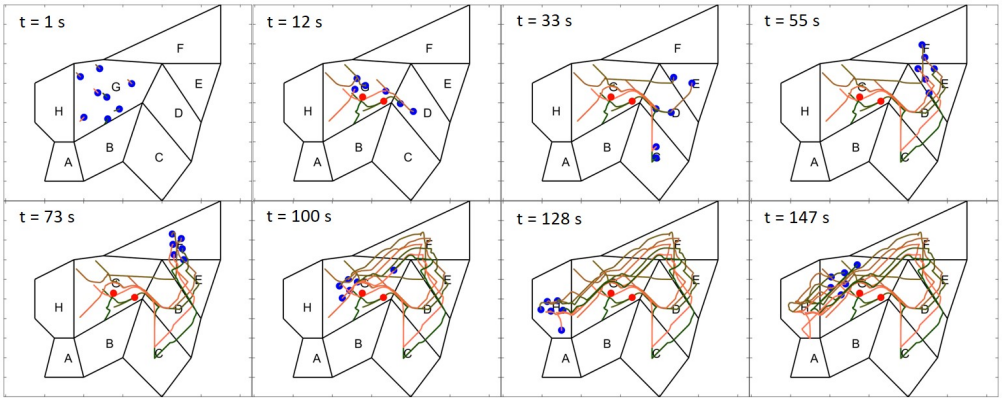
The task specification are expressed as $\phi_4 = \varphi_4^{\mu} \wedge \phi_{41} \wedge \phi_{42} \wedge \phi_{43}$, where φ_4^{μ} indicates a microscopic specification. First, from all the specifications and the region graph, we obtain a centralized symbolic plan $\mathscr{T}_4$: $(\pi_G \rightarrow \pi_G \wedge \pi_D \rightarrow \pi_C \wedge \pi_D \wedge \pi_E \rightarrow \pi_D \wedge \pi_E \wedge \pi_F \rightarrow \pi_E \wedge \pi_F \rightarrow \pi_F \rightarrow \pi_G \rightarrow \pi_H \rightarrow \pi_A \wedge \pi_H \rightarrow \pi_G \wedge \pi_H)^{\omega}$. We partition the centralized symbolic plan $\mathscr{T}_4$ into 3 different decentralized plans that can be executed by individual robots:

- $\mathscr{T}_{41}$: $(\dot{\pi}_G \rightarrow \pi_D \rightarrow \dot{\pi}_E \rightarrow \pi_F \rightarrow \pi_F \rightarrow \pi_F \rightarrow \pi_G \rightarrow \pi_H \rightarrow \dot{\pi}_H \rightarrow \pi_G)^{\omega}$,
- $\mathscr{T}_{42}$: $(\dot{\pi}_G \rightarrow \pi_D \rightarrow \dot{\pi}_D \rightarrow \pi_E \rightarrow \pi_F \rightarrow \pi_F \rightarrow \pi_G \rightarrow \pi_H \rightarrow \dot{\pi}_H \rightarrow \pi_G)^{\omega}$,
- $\mathscr{T}_{43}$: $(\dot{\pi}_G \rightarrow \pi_D \rightarrow \dot{\pi}_C \rightarrow \pi_G \rightarrow \pi_E \rightarrow \pi_F \rightarrow \pi_G \rightarrow \pi_H \rightarrow \dot{\pi}_A \rightarrow \pi_H)^{\omega}$.

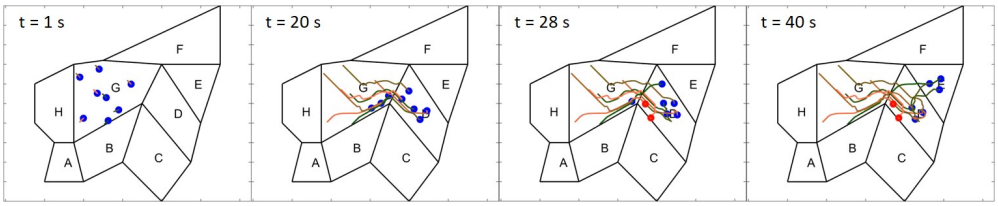
The symbolic plans loop back to the initial state after visiting the last state and the three groups synchronize at states denoted with a dot.

We implement both centralized and decentralized symbolic plans in MATLAB. We assume robots stop moving if they fail to receive velocity commands. The system can detect such failure and regard the static robots as obstacles. During the simulation, we manually send 0 velocities to 2 robots to simulate the failure. The executions with failures are shown in Fig. 2.12a and Fig. 2.12b.

We run all simulations in MatLab R2017b on a laptop with an Intel Core i7-6700HQ CPU@2.60GHz and 16GB RAM. We set the simulated robots to update velocities every $0.2s$. We record average computation time to obtain the velocities, as is shown in table 2.1, in each time period for both the centralized and decentralized execution. In the centralized execution, the computation time is the time to complete the dynamic assignment and compute all velocities, while in the decentralized execution, the computation time is only for computing the velocities. For continuous control synthesis, We use centralized



(a) Centralized execution



(b) Decentralized execution

Figure 2.12: Screenshots for 8 simulated robots performing a task given the same specifications, workspace, and initial conditions by executing the (a) centralized and (b) decentralized symbolic plans in Example 2.6.2. Blue dots represent working robots while red dots indicate robots that stop moving. The curved lines show the trajectories for all robots.

CBF in the execution of centralized and decentralized symbolic plans, in both Example 2.6.1 and 2.6.2.

Results: Example 2.6.1 shows the difference in task efficiency between the centralized and decentralized execution. The task requires swarm robots to occupy three regions at a time, and the centralized symbolic plan is partitioned into three decentralized programs. As there is no geometrical information about regions in the region graph, the synthesis and partition results in decentralized approach would make each sub-swarm (robots with the same decentralized plan) travel an unnecessarily long distance in the workspace. As is shown in Fig. 2.11b, one sub-swarm first visits region I, then goes up to C, and finally moves down to H because regions C, H, and I are equivalent topo-

logically. Similar transitions also happen in the other two sub-swarms where they travel towards the farthest goal regions for reaching the temporal subgoals. However, to create the required collective behaviors, the robots could choose a more efficient way by centralized coordination. As is shown in Fig. 2.11a, the dynamic assignment leads to swarm robots visiting neighboring regions in sequence, avoiding long-distance travel and trajectory intersections but still satisfying the specifications. The results in table 2.1 show that for 10 sets of different initial positions in region A, the maximum distance that one single robot travels and total distances of all robots decreases by 45% and 44%, respectively, after switching from decentralized to centralized execution. The execution time for one cycle of the task in the centralized execution is 46% less than that in decentralized execution, which means that the centralized coordination makes the system complete the task faster. However, the execution of the decentralized symbolic plans has an advantage of less computation time during each time step, as the symbolic plans are distributed before the execution in the decentralized approach while in centralized approach we solve an online IP during the execution.

Table 2.1: Comparison of the centralized and decentralized executions in Example 2.6.1. We show the average values of 10 trials with different initial positions of 15 robots. Max Distance is the maximum distance that a single robot travels. Total Distance is the sum of distances that all robots travel. Execution Time is the total time for all robots to complete one cycle of the specified task, where the period of updating control inputs is 0.2 s. Computation Time is the time to compute velocities of all robots in one period.

Ave.	Centralized	Decentralized
Max Distance (m)	22.34	40.95
Total Distance (m)	323.13	577.73
Execution Time (s)	111.72	207.68
Computation Time (s)	0.033	0.021

Example 2.6.2 presents the different behaviors given the same tasks and initial positions where an unexpected failure happens. The evolution of trajectories with different plan execution are shown in Fig. 2.12. In the execution of centralized symbolic plans

(Fig. 2.12a), when 4 robots go towards region D at $t = 12s$, two robots (red) fail to receive any velocity commands and they stop to move. With the dynamic assignment, the other working robots automatically "take" the roles of the failed robots, occupy region C, D, and E, and then continue the task successfully. However, in the execution of decentralized symbolic plans, when two robots fail and stop at $t = 28s$, two subswarms synchronize in region E and D, but no robots can occupy C, as the program for each robot is assigned before the execution. Thus, the whole system is unable to make progress due to the individual failures and cannot complete the specified task. The results show that the centralized execution with dynamic assigning online is more resilient to possible individual failures in swarm robotic systems, although it cannot guarantee that the system would recover from any failure, for example, in an extreme case when all robots fail to move. More details are in the submitted video.

2.6.3 Demonstration on Robotarium

Example 2.6.3. We show this example on the Robotarium platform, where we compare different behaviors of executing the centralized and decentralized symbolic plans with the same task specifications and initial conditions. Consider two working zones connected by a long corridor shown in Fig. 2.6c. The swarm, initially located in region E, is required to repeatedly only occupy region E ($\phi_{51} = \square\lozenge(\pi_E \wedge \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{E\}} \neg\pi_{\mathbf{r}})$), regions G and I simultaneously ($\phi_{52} = \square\lozenge(\pi_G \wedge \pi_I \wedge \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{G,I\}} \neg\pi_{\mathbf{r}})$), and regions B, D, and H simultaneously ($\phi_{53} = \square\lozenge(\pi_B \wedge \pi_D \wedge \pi_H \wedge \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{B,D,H\}} \neg\pi_{\mathbf{r}})$). The total specification is defined as $\varphi_5 = \bigwedge_{i=1}^{3} \phi_{5i}$. We first synthesize a centralized symbolic plan:

- $\mathscr{T}_5$: $(\pi_E \to \pi_C \wedge \pi_F \to \pi_B \wedge \pi_D \wedge \pi_H \to \pi_C \wedge \pi_F \to \pi_E \wedge \pi_G \to \pi_G \wedge \pi_I \to \pi_F)^{\omega}$.

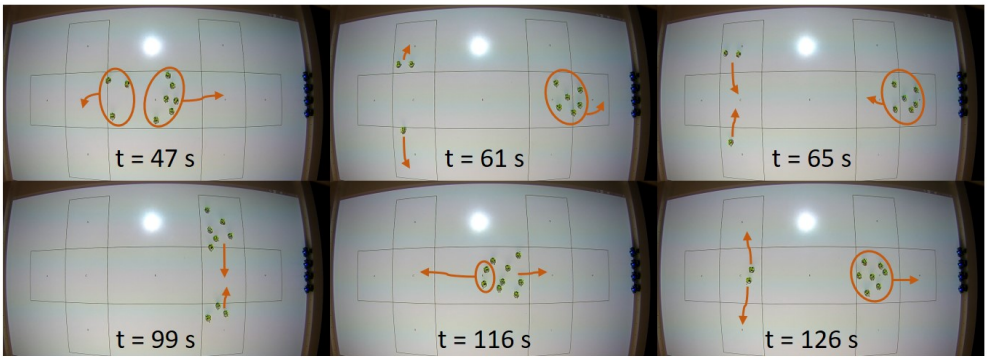
The symbolic plan loops back to the initial state after visiting the last state. Then, we

synthesize three decentralized plans $\mathscr{T}_{51}, \mathscr{T}_{52}, \mathscr{T}_{53}$ with the following runs and implement the symbolic plans on three groups of robots:

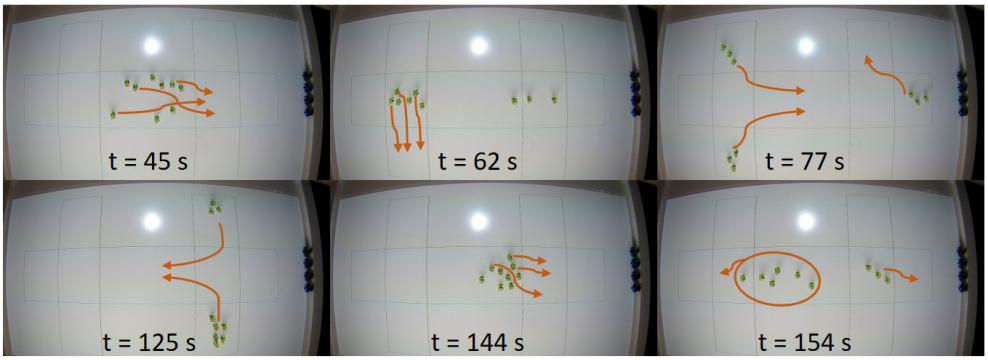
- $\mathscr{T}_{51}$: $(\dot{\pi}_E \to \pi_F \to \dot{\pi}_H \to \pi_F \to \pi_G \to \pi_G \to \dot{\pi}_G \to \pi_F)^\omega$,
- $\mathscr{T}_{52}$: $(\dot{\pi}_E \to \pi_C \to \dot{\pi}_B \to \pi_C \to \pi_E \to \pi_F \to \dot{\pi}_I \to \pi_F)^\omega$,
- $\mathscr{T}_{53}$: $(\dot{\pi}_E \to \pi_C \to \dot{\pi}_D \to \pi_C \to \pi_E \to \pi_F \to \dot{\pi}_I \to \pi_F)^\omega$,.

As before, the symbolic plans loop back to their initial state after visiting their last state and they synchronize with each other in the regions marked with a dot. We use decentralized CBF for velocity computation in the execution of both the centralized and decentralized symbolic plans.

Results: Figure 2.13 shows the implementation of Example 2.6.3 on the Robotarium platform. We compare the execution of the centralized and decentralized symbolic plans using the same task specifications and initial conditions. We note that in the centralized execution, robots spread out to occupy regions on the opposite side of the workspace in a more efficient way, where the robots chose their nearest neighbors and they did not have any crossing motion through another group of robots. In addition, the centralized approach only sent 2 robots to the left side of the workspace because only 2 regions need to be occupied simultaneously, which reduces the total distances that all robots travel. However, during the execution of the decentralized symbolic plans, robots move slowly to go across a group of other robots and most robots travel far distances to satisfy the decentralized symbolic plans. The execution of the decentralized symbolic plans in the same example needs 40% more time to complete one cycle of the repetitive task. More details are in the submitted video.



(a) Centralized execution



(b) Decentralized execution

Figure 2.13: Example 2.6.3. Robots are divided into three groups. Lines represent walls in the workspace and arrows indicate the motion.

2.7 Conclusions

In this work, we study the centralized and decentralized control of swarm robotic systems at both the symbolic and continuous levels. At the continuous level, we compare the centralized and decentralized versions of control barrier functions and demonstrate different behaviors when individual failure happens. The results show that decentralized control barrier functions guarantee collision-free motions even with individual failures while centralized computation might cause collisions. At the symbolic level, we use the synthesis framework in [55, 57] to obtain centralized and decentralized symbolic plans, and we propose a dynamic assigning method to make swarm robots execute the

centralized symbolic plans correctly. We present case studies in simulations and on physical robots to show different behaviors in centralized and decentralized execution. The results show that the execution of the centralized symbolic plan is more efficient in completing the task and more resilient to possible failures as it provides optimal coordination, while the execution of the decentralized symbolic plans is more efficient in computation as all plans are already assigned to robots before the execution. Our future work is to consider communications among robots in decentralized execution to enhance the coordination of swarm systems and improve the efficiency with decentralized control.

CHAPTER 3

AUTOMATIC CONTROL SYNTHESIS FOR SWARM ROBOTS FROM FORMATION AND LOCATION-BASED HIGH-LEVEL SPECIFICATIONS

3.1 Introduction

Swarm and multi-robot systems can perform various tasks such as surveillance [78], warehouse logistics [30], and object transport [20]. Such tasks might require robots to achieve certain formations for communication and collaboration, or even for entertainment [4].

Control laws that cause swarms to create a specific shape are referred to as formation control [63]. Many aspects of formation control have been studied, such as stability [25] and decentralized coordination [2]. However, work in formation control typically does not consider how to create control that ensures a formation happens in a specific location in the workspace, nor has past work enabled users to specify complex behaviors that include multiple different shapes, for different subswarms, in different locations, and automatically generate the appropriate control.

In this work, we create abstractions for formations and locations of swarms, and utilize the abstractions to synthesize compositional and flexible behaviors in a provably-correct way from user specifications. The framework we describe could impact a variety of application domains from digital agriculture where a farmer may deploy a swarm of UAVs for monitoring and spraying behaviors, to entertainment, where a robotics-novice designer might create complex formations, to construction where a civil engineer may deploy swarms as scaffolding for different structures.

This work leverages our previous work [57, 17], where [57] focuses on synthesizing

decentralized symbolic plans given high-level specifications, and [17] creates continuous control guaranteed to avoid collision, physically implement the symbolic plans, and mitigate deadlocks. Expanding on [57, 17] that only consider region-based specifications, in this work we create richer abstractions and specifications corresponding to both swarm formation and location. In addition, we design continuous controllers that are parametric in the specified formation and swarm size, and use a centralized assignment mechanism to ensure the execution of the symbolic plan satisfies the user-provided specification and avoids collisions.

We consider this work as addressing high-level behaviors for swarms even though our solution is centralized and may be considered as multi-robot. This is because our abstractions, specification formalism, and automatic control synthesis are agnostic to the exact number of robots which may change during execution, a characteristic of swarms.

Assumptions: In this work, we assume the environment and robot poses are known. Furthermore, we synthesize controls and provide guarantees for the robots assuming that they are holonomic; when controlling physical, differential drive robots, we increase the safety distance to mitigate the effects of the dynamic constraints, as described in Section 3.8.

This work presents (i) a novel abstraction and grammar that allow a user to specify location and formation-based swarm behaviors, (ii) automated correct-by-construction control synthesis for the swarm robots that guarantees that the task will be satisfied, if feasible, and (iii) demonstrations on simulated and physical swarm platforms that showcase the generality of the approach.

3.2 Related Work

Formation control has been studied extensively in swarm and multi-robot systems research [60]. Some work develops continuous control laws based on different robot sensing and interaction capabilities [60]: for example, a camera-based cooperative control framework for nonholonomic robots [23] and a position-based method with dynamic communication topology [76], while other works utilize graph theory to convert multi-robot formation control to a discrete path planning problem [21], such as robots forming lattice patterns [82] and switching formations on grids [38, 72]. Our work is different in that we tackle high-level behaviors that include both the sequencing of formations but also the automatic distributions of the robots into several sub-swarm formation, to achieve the desired high-level task in the continuous space.

Recently, compositions of swarm behaviors have been studied [68, 67]. The work in [68] focuses on continuous control for reaching certain spatial configurations and switching between different behaviors to guarantee effective information flow. The work in [67] focuses on selecting an optimal sequence of behaviors to achieve a certain task. In this work, we focus on creating abstractions and synthesizing controls from user-defined formations and temporal logic specifications, which automates the mission planning and sub-swarm assignment process compared to [68, 67].

Control synthesis from high-level specifications has been applied to multi-robot systems and swarms [75, 77, 91, 80, 31, 79]. For example, [75] deals with asynchronous motions of robots in high-level mission planning to guarantee collision and deadlock avoidance, and [77] composes motion primitives based on high-level specifications to maintain spatial configurations on a grid. In [91], the authors develop specifications for swarm robots and formally prove the system properties. Authors in [80] developed a

framework to automatically generate optimal behaviors for heterogeneous multi-robot teams to achieve missions specified in temporal logic, and [31] describes a method to generate a sequence of multi-robot policies for task allocation and planning from LTL specifications. In [79], the authors present a mechanism for a team of robots to coordinate to fulfill tasks given in LTL under uncertainty. Our work also considers task allocation and coordination during task execution, but different from [80, 31, 79] where each robot has its own specification, we assign temporal goals to an arbitrary number of robots under formation-dependent constraints and synthesize controls to achieve a global task specified in LTL.

3.3 Preliminaries

3.3.1 Linear Temporal Logic

Linear temporal logic (LTL) [29] is a formal language consisting of propositions and logical and temporal operators. Let AP be a set of atomic propositions. The syntax of LTL is defined as follows:

$$\varphi ::= \pi \,|\, \neg\varphi \,|\, \varphi \vee \varphi \,|\, \bigcirc \varphi \,|\, \varphi \,\mathscr{U}\, \varphi$$

where $\pi \in AP$ is a proposition, $\neg$ is negation, $\vee$ is disjunction, $\bigcirc$ is next and $\mathscr{U}$ is until. Other logical operators such as conjunction ($\wedge$), implication ($\Rightarrow$) and temporal operators such as *always* ($\Box$) and *eventually* ($\Diamond$) can be derived from these basic operators. An LTL formula over propositions AP is interpreted over infinite words $w \in (2^{AP})^{\omega}$. The language of an LTL formula φ, denoted by $\mathscr{L}(\varphi)$, is the set of infinite words that satisfy φ, i.e., $\mathscr{L}(\varphi) = \{w \in (2^{AP})^{\omega} \mid w \models \varphi\}$. Intuitively, $\Box\pi$ means proposition π has to be true at all times, $\Diamond\pi$ means proposition π will be true at some point, and $\bigcirc\pi$ means

that proposition π has to be true in the next step. We refer the readers to [29] for more details about LTL.

3.3.2 Abstractions and Specifications for Swarms

In [54, 17, 57] we propose a region based abstraction for swarm behaviors. We partition a 2D workspace into a set of regions $\mathbf{R} = \{r_1, ..., r_m\}$, and use a region graph $\mathbf{G_R} = (\mathbf{R}, \mathbf{E})$ to represent the connectivity of the regions, where $\mathbf{E} \subseteq \mathbf{R} \times \mathbf{R}$ represents possible transitions between regions, i.e., $(r_i, r_j) \in \mathbf{E}$ indicates robots can move directly from region r_i to r_j without going through any other region. We define $\pi_r \in AP$ as an atomic proposition which is true *iff* at least one robot is in region $r \in \mathbf{R}$. By using these propositions, a user can write location-based specifications in the following format:

$$\varphi = \phi^i \wedge \bigwedge_j \phi_j^s \wedge \bigwedge_k \phi_k^g, \tag{3.1}$$

where ϕ^i is a Boolean formula over AP representing the initial condition, $\phi_j^s = \Box\varphi_j^s$ are *safety* constraints where φ_j^s is a Boolean formula over $AP \bigcup \bigcirc AP$ representing conditions that must always hold, and $\phi_k^g = \Box\Diamond\varphi_k^g$ are system *livenesses* where φ_k^g is a Boolean formula over AP representing goals that must eventually be reached. For example, a formula $\varphi = r_1 \wedge \Box\neg r_2 \wedge \Box\Diamond r_m$ specifies that the swarm initially starts at region r_1, should repeatably visit r_m, and never enter r_2. Formula (3.1) has similar structure to *generalized reactivity (1)* (GR(1)) [12] which is a fragment of LTL that has polynomial time complexity in synthesis. However, different from GR(1), formula (3.1) does not contain environment propositions as the environment is assumed static and known.

3.3.3 Control Synthesis for Swarms

In our previous work [54, 17, 57], we proposed a control synthesis framework to satisfy specifications of swarm navigation tasks. We first apply LTL synthesis to obtain a symbolic plan given the high-level specifications. The symbolic plan $\mathscr{S} = \{q_0 q_1 q_2 ...\}$ can be represented as a sequence of states q_i such that each state q_i is labeled with a set of propositions $L(q_i) \subseteq AP$ that are true in state q_i. Then, we partition the symbolic plan into decentralized plans and execute the plans on different sub-swarms. By using control barrier functions (CBF) [89], we generate continuous controls that drive robots to reach the goal in collision-free paths. The synthesis framework guarantees that the swarm executes the task while avoiding collisions and satisfies the specifications, when possible, although there may be instances of deadlock which we mitigate.

3.4 Problem Formulation and Approach

This chapter solves the following problem: Given a workspace that is partitioned into a set of regions, a set of 2D/3D shapes, a robot swarm and formation (shapes) and location (regions) based high-level tasks, we automatically synthesize controls for all individual robots such that the robots correctly and safely satisfy the task. We assume the robots have full state information, i.e., their pose is known, and that they are velocity controlled.

To solve this problem we first create abstractions for formation and location-based robot behaviors, and then allow users to specify tasks over those abstractions using the LTL formulas in Eq. (3.1). We automatically synthesize a symbolic plan by leveraging the work in [57], and we use integer programming (IP) to assign robots to sub-swarms for achieving the symbolic plan. We automatically create continuous control for individ-

ual robots that implements the symbolic plan while guaranteeing collision avoidance.

3.5 High-level Specifications and Symbolic Synthesis

In this section we define the abstractions over which we create propositions that we use for the task specifications, and describe the specifications and the symbolic synthesis.

3.5.1 Abstractions: symbols and their physical grounding

Our abstraction contains three types of symbols; *location*, *target*, and *formation*, that allow a user to capture formation and location-based swarm behaviors.

Location symbols

As done in [54, 17], we partition the continuous workspace into regions and create a symbol for each region $\mathbf{R} = \{r_1, ..., r_m\}$. These symbols are grounded to the physical space and when used as propositions (Section 3.5.2), the proposition are true only if there is at least one robot in the corresponding region. The physical space and regions can be either 2D or 3D, as shown in the following.

Target symbol

We define the symbol T that represents existence of a target in a region. In section 3.5.2 we use this symbol in conjunction with formation symbols to create propositions whose truth values depend on whether the formation surrounds the target.

Formation symbols

We define formation symbols as a set $\mathbf{F} = \{C, F_1, ..., F_k\}$ where C represents coverage (robots distribute in a region according to a given coverage metric as described in Section 3.6.3), and $F_i (i \in \{1, ..., k\})$ is a shape formation.

We allow two types of shape formation in this paper: i) forming a perimeter of a 2D polygon or ii) surface/curve coverage of a 3D shape. For 2D shapes, a formation symbol F_i represents a polygon (a list of vertices) $s \in \mathbf{S}$ where

$$\mathbf{S} = \{[(x_1, y_1), ..., (x_p, y_p)] | x_i, y_i \in \mathbb{R}, p \geq 3\}.$$

For example, we can use $s_{sqr} = [(0,0),(1,0),(1,1),(0,1)]$ and $s_{tri} = [(1,0),(0,\sqrt{3}),(-1,0)]$ to represent a square and an equilateral triangle respectively. Note that the coordinates are only used to determine the shape rather than the actual formation size which is scaled based on the region.

For 3D shapes, the formation symbol F_i corresponds to a parametric function $\mathbf{p} : \mathbb{R} \rightarrow \mathbb{R}^3$ representing a 3D shape/curve. For example, $\mathbf{p}(t) = [rcos(t), rsin(t), ct]$ $(0 \leq t \leq 6c\pi)$ represents a helix with a radius of r and height of $6c\pi$. In this paper, we use the set of formation symbols $\mathbf{F} = \{C, sqr, tri, dia, hex, oct, pen, helix, sph\}$, where the symbols represent coverage, a square, a triangle, a diamond, a hexagon, an octagon, a five-point star, a helix (3D), and a sphere (3D) respectively.

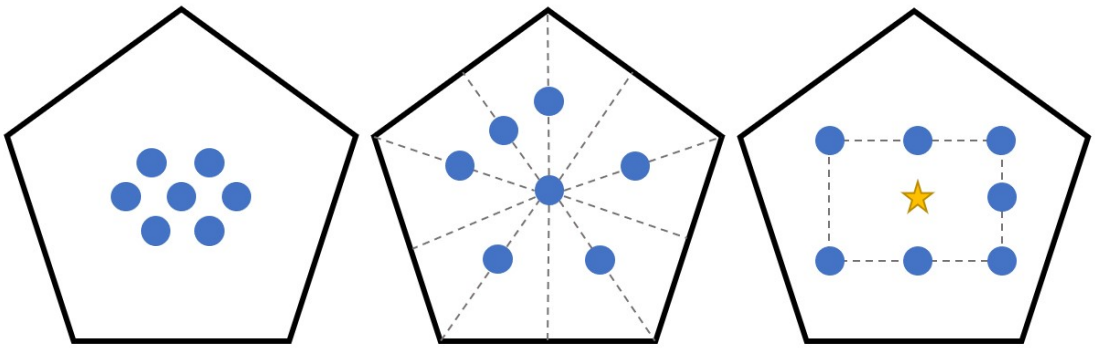
3.5.2 Proposition Creation

Given the symbols defined in section 3.5.1, we define the set of atomic proposition over which the specifications will be created, as $\mathbf{Prop} = (\mathbf{F} \cup \{\emptyset\}) \times \mathbf{R} \times \{T, \emptyset\}$, where $\times$ is the Cartesian product. We use the notation $F_j\_r_i\_T$ for the tuple (F_j, r_i, T), C_r_i for both

$(C, r_i, \emptyset)$ and (C, r_i, T) (same semantics), $F_j\_r_i$ for $(F_j, R, \emptyset)$, and r_i for both $(\emptyset, r_i, \emptyset)$ and $(\emptyset, r_i, T)$ (same semantics), where $F_j \in \mathbf{F}$ and $r_i \in \mathbf{R}$. The semantics of the propositions are defined as:

- $r_i = true$ indicates that at least one robot is in region r_i, regardless of formation.
- $C\_r_i = true$ indicates that at least one robot spreads out to cover the region represented by r_i. We note that "at least one robot" is the semantic definition due to r_i, but in practice a sub-swarm will cover the whole region.
- $F_j\_r_i\_T = true$ indicates that robots form the shape F_j around the target in region r_i with the target inside the shape, and $F_j\_r_i = true$ indicates that robots form the shape F_j in region r_i disregarding the location of the target, even if one exists.

Note that the proposition $F_j\_r_i\_T = true$ requires that there exists a target in region r_i, which might not be the case. Thus, we relax the semantics by assigning $F_j\_r_i\_T = true$ when robots form shape F_j anywhere in r_i if there is no target in region r_i. Figure 3.1 depicts possible physical representations of different atomic propositions.



(a) A swarm satisfying r_i. (b) A swarm satisfying C_r_i (c) A swarm satisfying $rect\_r_i\_T$

Figure 3.1: Proposition satisfying formations for seven robots in a pentagonal region. Blue dots represent locations of robots. The yellow star represents the target in the region.

3.5.3 Specifications

Given a set of propositions, **Prop**, users can write specification of the form (3.1). Given such a formula, we automatically reduce the number of propositions for synthesis and add formulas representing the swarm motion constraints, as described below.

Minimizing the set of propositions

Given the set of propositions that appear in the user's specifications, $\mathbf{Prop}_{spec}$, we create the set $\mathbf{Prop}' = \mathbf{Prop}_{spec} \bigcup \mathbf{R} \subseteq \mathbf{Prop}$, that will be used in the synthesis process. Calculating $\mathbf{Prop}'$ can reduce the number of propositions, which makes the synthesis faster as synthesis time grows exponentially with the number of propositions in the worst case [57].

Encoding motion constraints

In addition to the user specification, we automatically create formulas that represent motion constraints due to the topology and the formations.

For each region $r \in \mathbf{R}$, we first define sets of outgoing and incoming neighbors of r as $\mathbf{out}(r) = \{r' \in \mathbf{R} | (r, r') \in \mathbf{E}\}$ and $\mathbf{in}(r) = \{r' \in \mathbf{R} | (r', r) \in \mathbf{E}\}$, respectively. Then, we add for each r two *safety* formulas to the specifications: $\phi_{G1} = \square(r \rightarrow \bigcirc r \vee \bigvee_{r' \in \mathbf{out}(r)} \bigcirc r')$ and $\phi_{G2} = \square(\bigwedge_{r' \in \mathbf{in}(r)} \neg r' \wedge \neg r \rightarrow \bigcirc \neg r)$. The formula ϕ_{G1} indicates that robots in region r can only move to the outgoing neighbors of r or stay in r at the next step, and ϕ_{G2} indicates that robots cannot be in r at the next step if currently there is no robot located in r or any of its incoming neighbors.

To ensure formations in specific regions, we specify a *safety* formula $\phi_p =$

$\bigwedge_{F\_r\_T \in \mathbf{Prop}', F \in \mathbf{F}} \square(F\_r\_T \rightarrow (r \wedge \bigcirc r)) \wedge \bigwedge_{F\_r \in \mathbf{Prop}', F \in \mathbf{F}} \square(F\_r \rightarrow (r \wedge \bigcirc r))$ which encodes that if a sub-swarm achieves a formation in region r, there exist at least one robot in region r currently and at the next step. Intuitively, the safety formula connects formation and location to ensure that transition between formations is reachable in physical space given the environment topology.

Example: Given the workspace in Fig. 3.4 and the formation set **F** in section 3.5.1, the task is to: 1) repeatedly and visit r_1 and form a triangle in r_2 at the same time; 2) repeatedly form diamonds in r_1 and r_2 around the targets, and cover r_3 at the same time; and 3) never enter r_4. Using the symbols, the user can write the LTL formulas as:

- $\phi_{u1} = \square\lozenge(r_1 \wedge tri\_r_2)$
- $\phi_{u2} = \square\lozenge(dia\_r_1\_T \wedge dia\_r_2\_T \wedge C\_r_3)$
- $\phi_{u3} = \square\neg r_4$

Based on the task, $\mathbf{Prop}' = \{r_0, r_1, r_2, r_3, r_4, tri\_r_2,\ dia\_r_1\_T, dia\_r_2\_T, C\_r_3\}$. The added motion constraints are $\phi_p = \bigwedge_{j=1,\ldots,4} \phi_{pj}$:

- $\phi_{p1} = \square(tri\_r_2 \rightarrow (r_2 \wedge \bigcirc r_2))$
- $\phi_{p2} = \square(dia\_r_1\_T \rightarrow (r_1 \wedge \bigcirc r_1))$
- $\phi_{p3} = \square(dia\_r_2\_T \rightarrow (r_2 \wedge \bigcirc r_2))$
- $\phi_{p4} = \square(C\_r_3 \rightarrow (r_3 \wedge \bigcirc r_3))$.

The full specifications is

$$\Phi = \bigwedge_{i=1,2,3} \phi_{ui} \wedge \bigwedge_{i=1,2} \phi_{Gi} \wedge \phi_p.$$

Expressiveness: Given the abstractions, we can specify tasks that: 1) require robots to visit or avoid regions, and 2) require robots to form shapes in certain regions (at most one shape in each region), potentially around a target, or cover regions. Our abstractions currently do not allow multiple formations in one region or a formation that is created across multiple regions, however, we note that the choice or workspace partition and formation shapes are up to the user.

3.5.4 Synthesis

Given a specification, we use *Slugs* [27] to generate the symbolic plan $\mathscr{S}$ and we leverage the work in [57] to verify that the swarm will exhibit correct continuous behavior when continuously implementing the symbolic plan. For the example in Section 3.5.3, synthesizing the symbolic plan took $0.655s$ and the plan included 6 states.

3.6 Continuous Control Synthesis and Execution

In this section we first briefly explain the continuous execution from a given a synthesized symbolic plan. Then, we formalize the integer programming (IP) based approach to dynamically assign robots to sub-swarms. Finally, we define the continuous control synthesis for each individual robot.

3.6.1 Continuous Execution

Given a synthesized symbolic plan $\mathscr{S}$ from Section 3.3.3, we describe the continuous execution of a transition from state q_i to q_{i+1}:

First, we obtain the desired formations (including region visit and coverage) and the goal regions from $L(q_{i+1})$. Then, we calculate the minimum and maximum number of robots needed to achieve each desired formation, and use an IP based approach (Section 3.6.2) to assign robots to sub-swarms. Finally we apply the continuous control (Section 3.6.3) with control barrier functions [89] to each sub-swarm to achieve the desired formations with no collisions.

3.6.2 Sub-swarm Assignment

For correct continuous implementation of the symbolic plan, we must assign robots to sub-swarms based on the required symbolic transitions. Our method is based on linear assignment [15], which deals with optimally assigning n items (robots) to n places (goals). The main difference with respect to [15] is that we add constraints to the IP formulation to ensure the correct execution of the symbolic plan and enforces the topological constraints of the space, i.e. robots can only move to their neighboring regions or stay in the same region.

For N robots in a workspace partitioned into n regions, we define an assignment matrix $M \in \mathbb{R}^{N\times n}$ as

$$[M]_{ij} = \begin{cases} 1 \text{ if robot } i \text{ is assigned to region } j \\ 0 \text{ otherwise.} \end{cases}$$

For each symbolic transition (q_i, q_{i+1}), and for each region appearing in $L(q_{i+1})$, we define $N^{min}_{L(q_{i+1}),j}$ and $N^{max}_{L(q_{i+1}),j}$ as the minimum and maximum number of robots required to be in r_j in state q_{i+1} for the desired formation. In the following we will use N^{min}_j and N^{max}_j to denote $N^{min}_{L(q_{i+1}),j}$ and $N^{max}_{L(q_{i+1}),j}$ for simplicity. Furthermore, we define C_{ij} as the cost of robot i moving into region j. We can obtain an optimal assignment strategy by

solving the following IP:

$$
\begin{aligned}
M = \underset{[M]_{ij}\in\{0,1\}}{\operatorname{argmin}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{n} M_{ij}C_{ij} \\
s.t. \quad & \sum_{j=1}^{n} M_{ij} = 1 \quad \forall 1 \le i \le N \\
& \sum_{i=1}^{N} M_{ij} \ge N_j^{min} \quad \forall 1 \le j \le n \\
& \sum_{i=1}^{N} M_{ij} \le N_j^{max} \quad \forall 1 \le j \le n \\
& M_{ij} = 0 \quad \text{if } (r_{k_i}, r_j) \notin \mathbf{E},
\end{aligned}
\tag{3.2}
$$

where r_{k_i} is the region where robot i is currently located. We use the estimated travel distance for robot i to enter region j as the cost C_{ij}, thus we are optimizing the sum of the distances the robot travel. The cost function can be easily changed to capture other metrics such as minimum time.

The resulting assignment from (3.2) ensures that robots only move to their neighboring regions and the sub-swarm size fits the corresponding behaviors as long as N_j^{min} and N_j^{max} are well defined.

Given D_s, the inter-robot safety distance, we automatically determine N_j^{min} and N_j^{max} as follows:

Visit and Coverage. For 2D region visit and coverage, we set $N_j^{min} = 1$. For computing N_j^{max}, we first estimate the maximum number as $N_{j\_e}^{max} = \frac{A(r_j)}{\beta A_R}$, where $A(r_j)$ is the area of r_j, A_R is the area covered by a single robot, and $\beta > 1$ is a parameter. $N_{j\_e}^{max}$ is an initial guess of N_j^{max} based on the ratio of the region area and the robot dimensions, and β can be defined based on the safety distance between robots. We verify the result $N_{j\_e}^{max}$ by finding the goals given $N_{j\_e}^{max}$ and r_j using the method in Section 3.6.3, and let $N_j^{max} = N_{j\_e}^{max}$ if there exists a solution, otherwise we replace $N_{j\_e}^{max}$ by $N_{j\_e}^{max} - 1$ and repeat the process until a solution for the goals is found. For 3D regions, we choose the

central plane of the region to calculate N_j^{min} and N_j^{max} as the controller we use for region visit and coverage is the same as in 2D.

Shape formation. For 2D polygon forming, we let $N_j^{min} = p$ where p is the number of vertices of the specified shape as defined in Section 3.5.1. We enforce robots to at least occupy all the vertices of a given shape. For the maximum number, we make $N_j^{max} = \sum[l_i/D_s]$ where l_i is the length of each edge and $[x] = max\{m \in \mathbb{Z} | m \leq x\}$. The intuition is to ensure that the distances between goal points are at least D_s.

The edge length l_i is determined based on the formation $\mathbf{S}$ defined in 3.5.1 and the actual region size once the user has created the formation symbols. We assume $min(l_i) > D_s$ to avoid unsafe occupancy at the vertices. For 3D curve/surface forming, we have the user specify N_j^{min} and N_j^{max} as there are no vertices that robots have to occupy, and we verify that minimum distance between generated goal points is larger than the safety distance given N_j^{max}.

Optimality and feasibility: The robot assignment happens once for each transition in the symbolic plan $\mathscr{S}$, thus the assignment optimizes one transition; the assignment is not necessarily optimal for the whole task. In addition, when the number of robots cannot satisfy the constraints in Eq. 3.2, the execution will terminate as the task cannot be correctly executed even though the high-level specifications are realizable. In [57] we discussed an IP-based method to calculate ahead of time the minimum number of robots for each region in each state of the symbolic plan, needed to guarantee the execution of location-based tasks. In the future we will apply the method to formation-based tasks to provide N_j^{min} and N_j^{max} as the assignment constraints.

Computation complexity: Integer programming is generally NP-complete [62]. However, when the constraints and objective functions are linear, the IP becomes an *integer*

linear programming and it can be solved in polynomial time. In the IP formulation (3.2), all the equality constraints can be converted to a combination of two inequalities constraints, and all the constraints as well as the objective function are linear. Thus, the assignment can be calculated in polynomial time with respect to the number of robots. Our formulation can easily optimizes other metrics, such as minimum time, at the expense of longer computation times.

3.6.3 Continuous Control

Given the symbolic plan and the assignment of robots to sub-swarms, we present the approach to automatically synthesizing continuous controls that ensure robots to achieve the specified behavior while avoiding collisions. Control barrier functions (CBF) with nonlinear dynamics constraints are presented in [6, 93] for collision avoidance. However, in this work we use a simple version of CBF [89, 17] for multi-robot systems assuming holonomic robots, and mitigate the effect of the robots' dynamics by increasing the safety distance D_s in the physical demonstrations.

For all symbolic transitions, we divide the control synthesis into two sub-problems: 1) determine goal points for robots in the sub-swarms, and 2) create controls that drive robots to the goals with no collision.

Region visit and coverage. To determine the goal points in any region r_j for the visit behavior, we first pick as a starting point the region centroid, and then increase the number of points around the centroid in a hexagonal lattice pattern (Fig. 3.1a). The growing distance is $d = \beta D_s$, where $\beta > 1$, to ensure collision-avoidance. We choose a hexagonal lattice as this pattern guarantees the same distance between any two neighbor points as shown in Fig. 3.1a. For region coverage, we first find line segments that

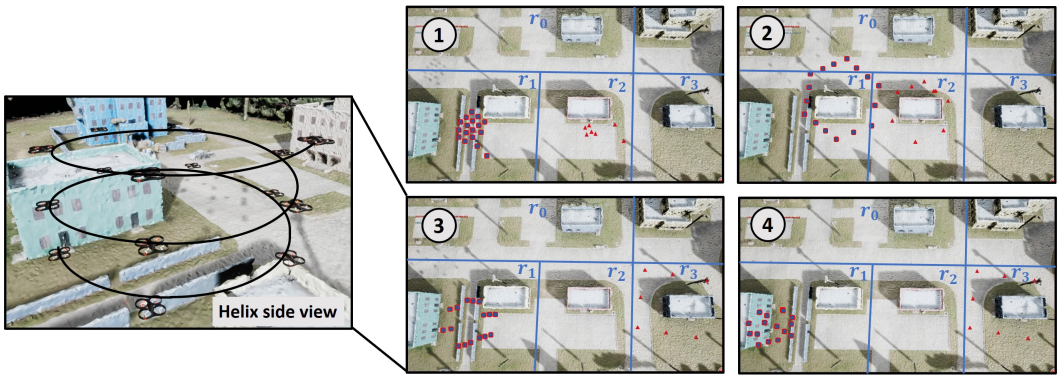


Figure 3.2: Screenshots from a demonstration of 16 UAVs and 8 rovers performing a high-level formation and location-based task in the AirSim simulation. Four target buildings are located in regions r_i ($i \in \{0,...,3\}$) respectively. The rounded squares and triangles represent the UAV and the rover locations respectively. The numbers 1-4 show the time sequence of the screenshots. A side view is provided for better visualization of a 3D helix.

connect the centroid with the vertices or the edge midpoints ($2p$ in total where p is the number of vertices), and divide these line segments evenly by N such that $(N-2)p+1 \leq N_R \leq (N-1)p+1$ where N_R is the number of robots. Then we choose the centroid and $N_R - 1$ points at the equidistant points on the line segments from the edges to the centroid. We check the distances d between points. If there exists a distance $d < D_s$, we replace N by $N+1$, thus creating more points to choose from, and repeat the checking process. If N becomes large enough such that distances between neighboring equidistant points are all less than D_s, the algorithm returns no solution. We note that here we assume convex regions; we can easily substitute any other coverage algorithm that would also work for non-convex regions.

To solve sub-problem 2), we run a linear assignment [15] to distribute goal points to individual robots, then design a PID controller for each single robot, and finally apply the CBF [89] to ensure that robots reach their goals with no collisions. CBFs are used to modify control inputs (velocities) such that robots are guaranteed to move with no collisions and reach their goals if there is no deadlock. For 3D regions, we choose the

vertically central 2D plane to execute region visit and coverage motions although we could also choose a 3D coverage metric.

Shape formation. Given the sub-swarm size N, we solve sub-problem 1) for a 2D polygon by making p robots occupy the shape vertices, and $N - p$ distribute evenly on the edges. For sub-problem 2), we design a two-step process for obtaining continuous controls, which first drives robots to surround the centroid of the desired formation or the target, depending on the required behavior, and then assign them to different goals to complete the desired shape. In the surrounding step, we compute the velocities based on robot poses with the objective of evenly distributing robots around the polygon centroid or target. After completion of the surrounding step, we assign goals to robots and apply a PID controller with the CBF to make the robots form the shape while avoiding collisions. Once all sub-swarms finish forming the shapes, they proceed to the next state in the symbolic plan. When forming 3D shapes, we use the parametric function to generate a list of goal points given N, the number of robots, to solve sub-problem 1). For sub-problem 2), instead of the two-step control process for 2D polygons, we assign robots to goal points and drive them towards their goals to achieve the 3D shape.

Deadlocks:. In multi-robot systems, deadlocks are situations where robots cannot make progress towards their goals. In [17] we discussed different deadlock scenarios and provided perturbation and roadmap-based method to mitigate deadlocks. In general, in environments with obstacles, we cannot guarantee deadlock avoidance; however, in practice, such techniques often result in deadlocks being resolved.

3.7 simulations

We first demonstrate the control framework on 16 UAVs and 8 ground vehicles in AirSim[81]. The simulation environment is a space with four target buildings. We partition the 3D workspace into r_{i_j} ($i \in \{0,1,2,3\}, j \in \{0,1,2\}$) such that the target buildings are in regions r_{i_0}, $i \in \{0,...,3\}$. The index j represents the height of the regions: $j = 0$ represents the ground-plane 2D regions, $j = 1$ represents 3D regions at 10 to 20 meters above ground, and $j = 2$ represents 20 to 30 meters above ground.

3.7.1 Specifications

Initially, the UAVs and the rovers are located in regions r_{1_1} and r_{2_0} respectively. We require the swarm to accomplish three goals: repeatedly form hexagons around the targets in r_{1_1} and r_{2_0}, repeatedly form a square around the target in r_{3_0} and a helix in r_{1_1}, and repeatedly form a square around the target in r_{3_0} and a sphere in r_{1_2}. Note that we assume the UAVs only fly in the air and rovers only move on the ground, which is encoded in the region graph. Formally, the specifications are expressed in LTL and include:

- $\square\lozenge(hex\_r_{2\_0\_}T \wedge hex\_r_{1\_1\_}T \wedge \bigwedge_{k\neq 2\_0,1\_1} \neg r_k)$,
- $\square\lozenge(sqr\_r_{3\_0\_}T \wedge helix\_r_{1\_1} \wedge \bigwedge_{k\neq 3\_0,1\_1} \neg r_k)$,
- $\square\lozenge(sqr\_r_{3\_0\_}T \wedge sph\_r_{1\_2} \wedge \bigwedge_{k\neq 3\_0,1\_2} \neg r_k)$.

The negation of propositions in the formulas indicates that no robots should be in those regions, thus robots should only appear in the regions that have the specified formations.

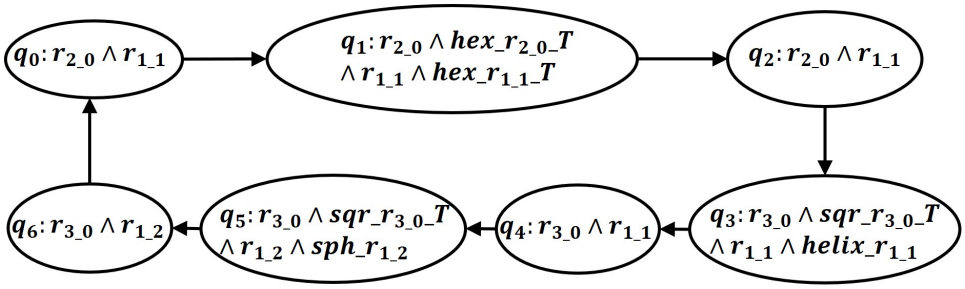


Figure 3.3: Symbolic plan for the example of Section 3.7

These specifications simulate tasks where UAVs and ground vehicles collaboratively guard some target buildings.

3.7.2 Results

From the given specifications in Section 3.7.1, we use the approach in Section 3.5 to synthesize a symbolic plan shown in Fig. 3.3, where states q_i ($i \in \{0, ..., 6\}$) are labeled with propositions that are true in that state.

We execute the synthesized symbolic plan on 16 UAVs and 8 rovers using the approaches in Section 3.6. The result shows that the swarm achieves the high-level task correctly and safely (Fig. 3.2). For executing two cycles of the symbolic plan, the average computation time for sub-swarm assignment is $0.08s$, and the average time for computing the velocity commands during execution is $0.12s$. The time for synthesizing the symbolic plan in this demonstration is $0.97s$. The simulation was run on a desktop machine with an Intel Core i7-7700 CPU@3.6GHz and 16GB RAM.

3.8 Physical Demonstrations

In this section we demonstrate our approach on physical robots, using 20 *Coachbot V2.0* robots [88] (customized differential-drive ground robots) in an environment partitioned into five regions. We use the formation symbols **F** (Section 3.5.1) for creating propositions and specifications.

3.8.1 Specifications

The swarm is initially in r_0. We require the swarm to achieve three goals: repeatedly form an octagon around the target in r_1 and cover the space of r_2 at the same time while avoiding all other regions, repeatedly form a diamond in r_3 and a five-point star in r_4 at the same time while avoiding all other regions, and repeatedly gather in r_0. We add a safety specification that prevents the swarm from being in r_1 and r_4 simultaneously at any time. The specifications is formally expressed in LTL (part of the full formula is shown here):

- $\square\lozenge(oct\_r_1\_T \wedge C\_r_2 \wedge \neg r_0 \wedge \neg r_3 \wedge \neg r_4)$
- $\square\lozenge(dia\_r_3 \wedge pen\_r_4 \wedge \neg r_0 \wedge \neg r_1 \wedge \neg r_2)$
- $\square\lozenge(r_0 \wedge \neg r_1 \wedge \neg r_2 \wedge \neg r_3 \wedge \neg r_4)$
- $\square\neg(r_1 \wedge r_4)$.

3.8.2 Results

We synthesize a symbolic plan from the given specifications in Section 3.8.1 and show the execution in Fig. 3.4. The robots start from r_0, first form an octagon in r_1 and cover r_2, then form a diamond and a five-point star in r_3 and r_4 respectively, and go back to gather in r_0, as shown in Fig. 3.4. All the objectives and constraints are satisfied using the symbolic and continuous control synthesis. Note that the robots gather in r_2 and r_3 before entering r_4 because robots are not allowed in r_1 and r_4 at the same time and the unsafe transitions between r_1, r_2 and r_3, r_4 are ruled out during the synthesis process.

3.8.3 Discussion

Controlling a swarm of physical robots correctly and safely is challenging. We address the problems that we have encountered during the implementation on physical robots and discuss the possible solutions that might improve the physical swarm ability to perform high-level tasks.

In contrast to the simulation, where the behavior of the swarm satisfied the desired behaviors and no collisions occurred, during the physical demonstrations, collisions did happen when too many robots were in a narrow space, e.g., when forming the five-point star in r_4 in Fig. 3.4. This is due to the mismatch between our assumptions of full state information and idealized kinematic robots and the reality of the physical platform where we have 1) imperfect localization, 2) nonlinear robot dynamics, and 3) a tight workspace. Specifically, we assume a honolomic kinematic model for each robot to design collision-free control, and then map the robot velocity in the global frame to the wheel speeds for coachbots. However, the mapping is not fully accurate as the differential-drive robots cannot move sideways, which might violate some constraints

in the CBF when robots get too close in a tight environment. Usually increasing the safety distance can improve the robustness for collision avoidance. In this work, we set the safety distance $D_s = 1.2D_r$ where D_r is the robot diameter, for the purpose of fitting 20 robots in one region to achieve the synthesized symbolic plan. Such tight safety margins, together with the nonlinear dynamics and sensing error, lead to some collisions in the physical demonstration. Our future work is to consider robot dynamics in the CBF formulation thus guaranteeing collision avoidance [28, 6, 93].

3.9 Conclusion and Future Work

In this work we propose a novel abstraction for robot swarms that allows a user to specify tasks regarding the swarm's formations and locations. We describe a framework for automatically creating control for the swarm robots such that the swarm is guaranteed to achieve its tasks while avoiding collisions with the environment and between the robots. We demonstrate the approach both in simulation and on physical robots.

In future work we will robustify the approach with respect to constrained physical systems. Specifically, we will create continuous controls by adding robot dynamics to the CBF constraints. In addition, we will study the amount of information that individual robots need, and design communication and synchronization frameworks for swarms to achieve such high-level tasks in a distributed manner to increase the scalability of the approach.

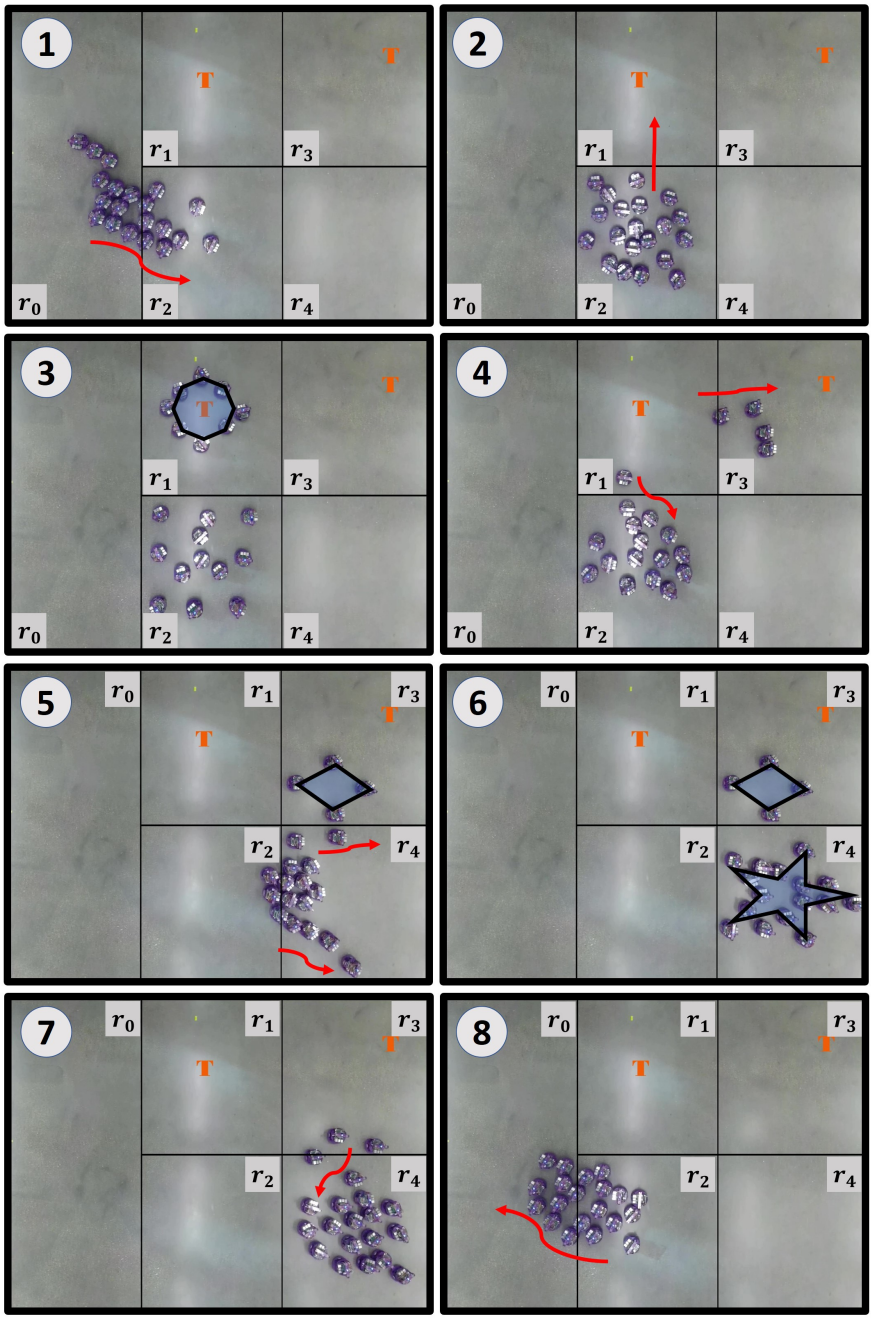


Figure 3.4: A demonstration of 20 coachbots performing a high-level task. Two targets (labeled with "T") are located in r_1 and r_3. The arrows indicate the motion of nearby robots.

CHAPTER 4

DISTRIBUTED CONTROL OF ROBOTIC SWARMS FROM REACTIVE HIGH-LEVEL SPECIFICATIONS

4.1 Introduction

Research in swarm robotics studies the coordination and interaction of large numbers of simple robots in terms of robustness, flexibility, and scalability [13]. Robot swarms have various potential applications such as surveillance [78], warehouse logistics [30], and collective construction [64]. While one direction of research on robot swarms focuses on creating simple single robot behaviors and analyzing the emergent behavior of the swarm [83, 46, 8] , recently researchers have been developing control schemes to achieve high-level swarm tasks in a top-down manner [42, 56, 34, 26]. These approaches use formal languages to specify global high-level tasks, and synthesize controls for individual robots such that the resulting swarm behaviors satisfy the specifications. However, these approaches either require centralized control for the robots [42, 56], or they lack reactivity to possible environment events [34, 26]. In this work, we propose a **distributed** control scheme for swarms to achieve **reactive high-level tasks**.

As a motivating example, consider the workspace partitioned into 5 rooms r_i, $i \in \{0,...,4\}$ shown in Fig. 5.1. There are two static targets in room r_3 and r_4. We assume that there is an environmental event "danger" that the swarm must react to; when there is no danger, the robots should form triangles in r_0, r_1, and r_2 at the same time but when danger is detected, the robots should form a square and a hexagon around the targets in r_3 and r_4.

Given such a reactive, globally-specified formation task, our objective is to auto-

matically synthesize controls for each individual robot such that the swarm completes the task correctly. Our previous work [16] proposed an abstraction for formations and presented a centralized control synthesis approach for accomplishing non-reactive high-level tasks given as linear temporal logic (LTL) formulas over the abstraction. However, in swarm applications, due to the scalability and communication costs, one prefers decentralized control, where robots only use their local information to make decisions, to achieve the global specifications. In this work, we present a novel control scheme, where we only synthesize a centralized symbolic plan offline based on the LTL specifications, but execute the symbolic plan in a decentralized manner by applying an auction-based algorithm to task assignment during transitions in the symbolic plan.

Assumptions. In this work, our abstraction allows at most one formation in a region at one time, as in [16]. Furthermore, we assume that the reactivity is non-instantaneous—swarms only detect and react to an environmental event after completing a transition in the symbolic plan. This means that the swarm will not change its behavior in the middle of a transition. We also assume that the map of the workspace is known and all robots can detect the environment events.

Contributions. This chapter presents (i) a distributed control method for executing a reactive finite automaton for swarm tasks, which satisfies robot number constraints. (ii) an IP-based approach for calculating constraints on swarm size, to ensure a priori feasibility of reactive high-level tasks. (iii) demonstrations of simulated and physical swarms to illustrate our approach.

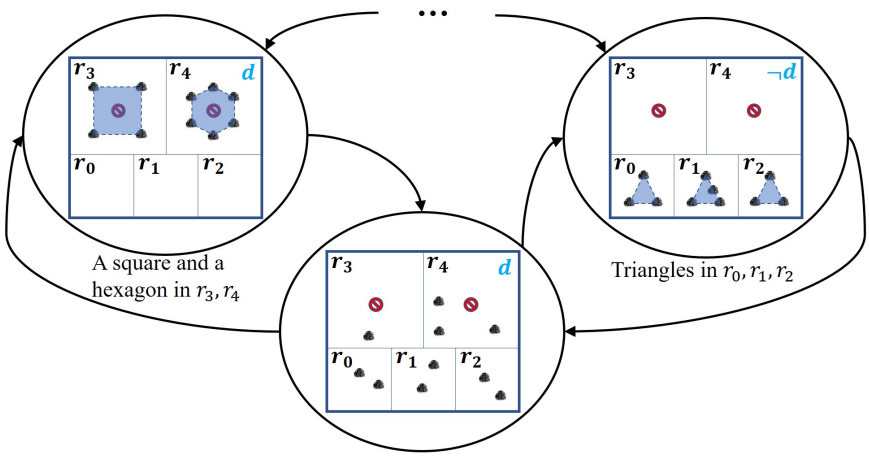


Figure 4.1: Part of a finite automaton representing a formation-based reactive task where d is the environment event.

4.2 Related work

Decentralized task assignment. There is a large body of work on decentralized task assignment for teams of robots (e.g. [22, 79, 73, 52]). Some researchers focus on the communication mechanism that enables the robots to satisfy task constraints: in [22], the authors propose an algorithm that guarantees the satisfaction of hard constraints and incorporates soft constraints into task allocation. Our work similarly uses auction-based methods, but we use our offline computation of sub-swarm size constraints to influence the online decentralized assignment. Other work focuses on dynamic assignment and planning during execution: [79] proposes a decentralized framework for multi-robot task assignment to satisfy LTL specifications under sensing uncertainties, and [73] shows a method using redundant robots for efficient task assignment with uncertain travel time. Unlike [79] that focuses on task allocation during single transitions in the policies, or [73] that requires redundant robots for a goal, our work creates strict sub-swarm size constraints that take into account the entire symbolic plan and applies them to online task allocation. Our work has offline planning using linear temporal logic synthesis, and

uses online task assignment that extends the auction-based algorithm in [52]. The main difference of the assignment algorithm is that [52] requires a fixed number of robots for each task while our algorithm allows a flexible number for each task and also provides guarantees of satisfying sub-swarm size constraints.

Reactive synthesis. Reactive synthesis [71] is the problem of generating a strategy such that for every environment behavior, the system has a behavior that will ensure a specification is satisfied. We leverage the work in [12] which presents reactive synthesis algorithms for a computationally more efficient fragment of LTL (GR(1)). Reactive synthesis has been applied to various robotics systems (e.g., humanoid robots [50] and manipulators [35]). Recently, reactive synthesis has been studied in the context of robotic swarms [56] in a centralized manner. In this work we create a decentralized execution of the symbolic plans generated by reactive synthesis.

Top-down control synthesis for swarms. The authors in [42] develop a hierarchical framework to create velocity commands for robots in a swarm from high-level specifications. Then, they construct the abstractions for individual robots and propose an algorithm to reduce inter-robot communication from the global specifications in [43]. Recently, [54, 58] presented a formal synthesis approach for creating decentralized symbolic plans from global LTL specifications for non-reactive swarm navigation problems, and we applied the synthesized symbolic plans to physical robots and studied the collision avoidance and deadlock mitigation during the continuous execution [17]. Then, in [16], we propose an abstraction and control synthesis framework that coordinates swarms to achieve non-reactive formation-based tasks in a centralized manner. Based on those synthesis approaches, this work focuses on decentralized execution of a finite automaton for reactive formation-based tasks. Our method is different from [54, 58] in that we address reactive tasks.

4.3 Prelimelaries

4.3.1 Abstractions for formation-based swarm tasks

In our previous work [16], we proposed abstractions that capture location and formation based swarm behaviors and created LTL specifications for desired swarm tasks. Intuitively, locations indicate regions of interest in the workspace, and formations are requirements on the relative distances between the robots. In [16], swarm formations create polygonal shapes where robots are distributed on the perimeters. The abstractions we defined contain three sets of symbols: location symbols $\mathbf{R} = \{r_1, ..., r_m\}$, formation symbols $\mathbf{F} = \{F_1, ..., F_k\}$, and a target symbol $\{G\}$ which indicates whether a point of interest exists in a region. We use these symbols to create an atomic proposition set $\mathbf{Prop} = (\mathbf{F} \cup \{\emptyset\}) \times \mathbf{R} \times \{G, \emptyset\}$, with the following semantics:

- $r_i = true$ indicates that at least one robot is in region r_i.
- $F_j\_r_i\_G = true$ indicates that robots form the shape F_j around the target G in region r_i, and $F_j\_r_i = true$ indicates that robots form the shape F_j in region r_i irrespective of the location or existence of the target,

where F_j is grounded to a 2D polygon (a list of vertices). For example, for the task partially depicted in Fig. 5.1, $\mathbf{R} = \{r_i | i \in \{0, ..., 4\}\}$ and $\mathbf{F} = \{T, S, H\}$ where T is a triangle, S is a square, and H is a hexagon. The propositions, T_r_0, T_r_1, and T_r_2 indicate robots forming triangles in r_0, r_1, and r_2, and $S\_r_3\_G$, $H\_r_4\_G$ indicate robots forming a square and a hexagon around targets in r_3 and r_4 respectively.

4.3.2 LTL Synthesis and Finite Automata

Given the propositions in Sec. 4.3.1, we specify a high-level task as a linear temporal logic (LTL) formula φ, and use the algorithm in [12] to create a (not necessarily unique) control automaton which, if executed, guarantees that the robots will satisfy φ. The generated automaton is a tuple $\mathbb{A} = (Q, Q_0, X, Y, \delta, L)$, where Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, X is the set of environment propositions, Y is the set of robot propositions, $\delta \subseteq Q \times Q$ is a transition relation, and $L : Q \to 2^{X \cup Y}$ is a labeling function which maps a state to a set of propositions that are true in that state. Revisiting the example in section 4.1, we synthesize an automaton $\mathbb{A}$ shown in Fig. 4.3a that satisfies the specification.

4.3.3 Market-based task assignment

Market-based approaches have been applied to decentralized task assignment for multi-robot systems [40]. The auction assigning method in this work is built on [52], where given n robots and m tasks, the authors proposed a distributed bidding framework to split the robots into groups of size n_k ($k \in \{1, ..., m\}$), where each task k requires exactly n_k robots, and $n = \sum_{k=1}^{m} n_k$. In their framework, each robot i ($i \in \{1, ..., n\}$) has a *selection function* $f_s(T)$ to select a task k_i and create a bid $b_i \geq 0$. In addition, each robot i has an estimation of the *availability* $\alpha_k^{[i]}$ ($\alpha_k^{[i]} \in \{0, ..., n_k\}$) of task k which represents the number of robots that can still be assigned to task k. Robots communicate with their neighbors, and the robot with the highest bid finalizes the task selection, while other robots repeat the process until all robots are assigned to tasks. The proposed auction method guarantees that exactly n_k robots are assigned to task k, and the final assignment can be reached after exploring at most $n(n+1)/2$ assignments.

4.4 Problem formulation and approach

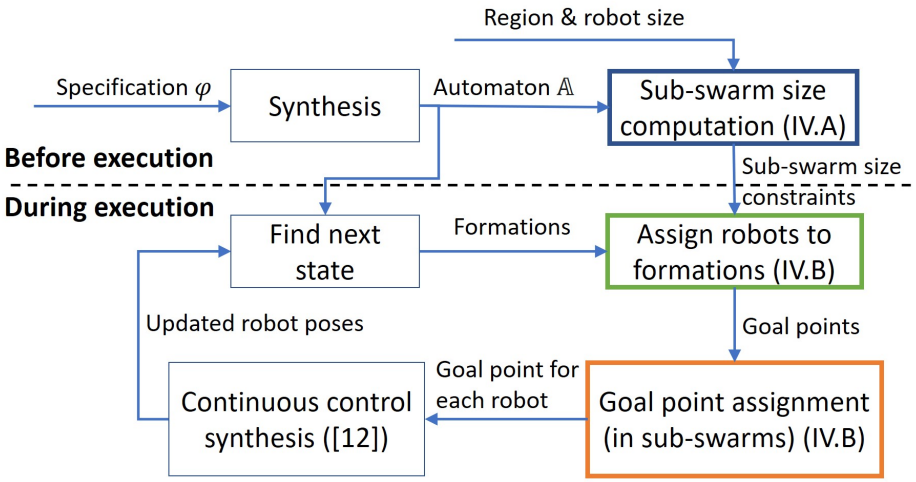


Figure 4.2: Flowchart of our control synthesis approach for high-level reactive swarm tasks. The contributions of this work are in the boxes with the thick border.

This work solves the following problem: Given n robots, a finite automaton $\mathbb{A} = (Q, Q_0, X, Y, \delta, L)$ synthesized from a high-level reactive swarm task φ, a 2D workspace consisting of polygonal regions $\mathbf{R}$ and targets, a set of possible 2D formations $\mathbf{F}$, and minimum and maximum numbers of robots needed for a formation F_i or a region r_j, we find, in a distributed manner, controls for the individual robots such that the swarm accomplishes the high-level task φ.

Our approach to the control synthesis problem is shown in Fig. 4.2. Before execution, users specify reactive, high-level tasks, from which we synthesize a finite automaton $\mathbb{A}$ [12]. We calculate constraints on the sub-swarm size for each transition in $\mathbb{A}$, based on the physical size of the regions and robots, and on the transitions in $\mathbb{A}$; these constraints are used during execution.

When executing the task, each robot has a copy of $\mathbb{A}$, and for every transition, it communicates with its neighbors to determine an assignment in a decentralized way. The assignment has two phases: assigning robots to sub-swarms, i.e. deciding which

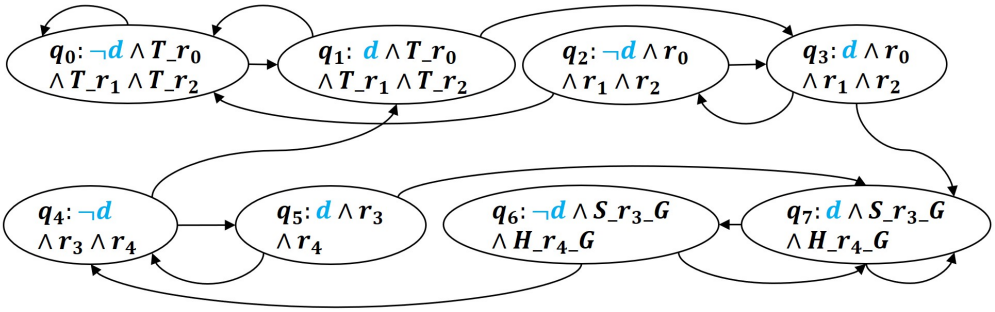
region they should go to and in which formation, and assigning them to goal points. Both phases use the same auction process described in Algorithm 2. Once every robot has a goal point, it synthesizes continuous control to reach the goal while avoiding collisions, as in [17]. Once all robots reach their goals, the process repeats; based on the environment state (the value of the propositions in X), the robots determine the required transition and proceed to determining sub-swarms and goal assignments followed by motion to the goal. We assume the swarm observes the environment state only once a transition has been completed, and the swarm contains enough robots to satisfy all constraints on the sub-swarm sizes—the total number of robots needed is known before execution after computing the constraints.

4.4.1 Computing sub-swarm size constraints

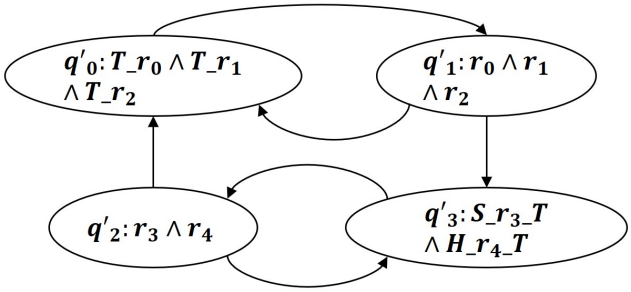
The LTL synthesis algorithm does not take into account the actual sub-swarm size because the specifications only capture the symbolic abstraction of the swarm motions. However, to physically execute the synthesized finite automaton, when distributing the robots into sub-swarms, we must make sure that sub-swarm size can satisfy the requirement of the corresponding behaviors. For example, we should assign at least 4 robots to form a square, or we cannot assign more robots than can safely move in a region.

Our previous work [16] myopically considered such constraints for the current transition; this might result in assignment failures when sub-swarm sizes cannot satisfy constraints on future transitions. For example, consider a transition that requires the swarm to be divided into two sub-swarms a and b. If the next transition requires sub-swarm a to be divided into three sub-swarms, then in the current swarm assignment, we must require that a contain at least three robots. In [16] we had no such constraints. In this

work, to eliminate future failures due to sub-swarm size, we calculate, before execution, constraints on sub-swarm size for all the transitions using integer-programming (IP). The proposed method builds on the quantitative analysis of swarm size in [58] which computes the minimum number of robots for non-reactive location-based tasks.



(a) Automaton $\mathbb{A}$ that satisfies the specifications in section 4.1



(b) A simplified automaton $\mathbb{A}'$ obtained from $\mathbb{A}$

Figure 4.3: (a) An automaton that implements the specifications in Section 4.1. (b) the simplified automaton $\mathbb{A}'$ used to calculate the robot number constraints for the transitions in $\mathbb{A}$.

Abstracting $\mathbb{A}$. For reactive tasks, the synthesized automaton $\mathbb{A}$, which is deterministic, usually contains many states and transitions, since for any environment behavior, there needs to be an appropriate reaction (system action). This may result in intractability of computing sub-swarm size constraints for each transition. However, we observe that only the robot propositions Y, i.e. the regions and formations, and not the environmental events, influence the constraints on sub-swarm size. Thus, to calculate the constraints on

the number of robots per transition, we first abstract $\mathbb{A}$ by a nondeterministic automaton $\mathbb{A}' = (Q', Y, \delta', L')$ where Q' is a set of states, Y is the same as in $\mathbb{A}$, δ' is the state transition function and L' is the labeling function such that $L'(q') \subseteq Y$ is the set of robot propositions that are true in state $q' \in Q'$. We create $\mathbb{A}'$, which has fewer transitions than $\mathbb{A}$, by grouping states in $\mathbb{A}$ with the same robot propositions Y and maintaining a mapping f between transitions in $\mathbb{A}'$ and transitions in $\mathbb{A}$. We then calculate bounds on the number of robots for transitions in $\mathbb{A}'$. When executing $\mathbb{A}$, we use the corresponding transitions in $\mathbb{A}'$ to determine the constraints on the robot sub-swarm size.

Algorithm 1: Abstracting $\mathbb{A}$

Input: $\mathbb{A} := (Q, Q_0, X, Y, \delta, L)$
Output: $\mathbb{A}' := (Q', Y, \delta', L')$, f

1: $Q' := \emptyset, \delta' := \emptyset$, $f := \emptyset$, $h := \emptyset$
2: **for** $q \in Q$ **do**
3: $y := L(q) \cap Y$
4: **if** $\exists u' \in Q'$ s.t. $L'(u') = y$ **then**
5: $q' := u'$
6: **else**
7: $q' := createNewState(y)$
8: $Q' := Q' \cup \{q'\}$
9: $L'(q') := y$
10: **end if**
11: $h := h \cup \{(q, q')\}$
12: **end for**
13: **for** $(q, p) \in \delta$ **do**
14: $f := f \cup \{((q,p),(q',p')) | (q,q') \in h \wedge (p,p') \in h\}$
15: **if** $(q', p') \notin \delta'$ **then**
16: $\delta' := \delta' \cup \{(q', p')\}$
17: **end if**
18: **end for**

We present the procedure for obtaining $\mathbb{A}' = (Q', Y, \delta', L')$ from $\mathbb{A} = (Q, Q_0, X, Y, \delta, L)$ in Algorithm 1. We first group states (Lines 2-12), and then map the transitions (Lines 13-18). To group states, we create a state q' for every unique set of system propositions Y that appear in Q (Lines 4-10). We create the mapping h to keep

track of which state in $\mathbb{A}'$ corresponds to states in $\mathbb{A}'$ (Line 11). To create f, that maps transitions in $\mathbb{A}$ to transitions in $\mathbb{A}'$, we find the corresponding states $q', p' \in Q'$ for each (q, p) in δ (Line 14). Finally, we create the transition set δ' (Lines 15-17). Since (q', p') have the same robot propositions (sub-swarm partition and formations) as (q, p), the calculation of the bounds on the sub-swarm size is equivalent to the direct computation on $\mathbb{A}$. For example, the automaton $\mathbb{A}$ synthesized from the specifications in section 4.1, shown in Fig. 4.3a, has eight states and 16 transitions. Using algorithm 1 to abstract $\mathbb{A}$ we get $\mathbb{A}'$ which only has four states and six transitions (Fig. 4.3b). Each transition in $\mathbb{A}'$ is mapped to some transition(s) in $\mathbb{A}$, e.g., (q'_0, q'_1) in $\mathbb{A}'$ is mapped to (q_1, q_3). We calculate the bounds on sub-swarm size for transitions in $\mathbb{A}'$, and map them back to $\mathbb{A}$ during execution. This reduces the number of variables and constraints in the IP, which increases the scalability of our approach.

IP formulation. Given the simplified automaton $\mathbb{A}' = (Q', Y, \delta', L')$, for any state $q' \in Q'$, we define the sets of previous and post states of q' as $\mathbf{pre}(q') = \{u' \in Q' | (u', q') \in \delta'\}$ and $\mathbf{post}(q') = \{v' \in Q' | (q', v') \in \delta'\}$. For each transition $(q', p') \in \delta'$, we define a set of non-negative integer variables $x^{q',p'}_{\mathbf{r}_{q'},\mathbf{r}_{p'}}$ where $\mathbf{r}_{q'}$ and $\mathbf{r}_{p'}$ represent the regions that are in the labels of state q' and p' respectively, such that they are adjacent in the workspace, i.e. robots can move from $\mathbf{r}_{q'}$ to $\mathbf{r}_{p'}$. For example, in Fig. 4.3b, $x^{q'_1,q'_3}_{r_0,r_3}$ represents the number of robots that are sent from r_0 to r_3 during the transition from q'_1 to q'_3. Then, we define two integers, $N^{q',min}_{r_i}$ and $N^{q',max}_{r_i}$, to represent the minimum and maximum numbers of robots that are allowed in region r_i in state q'. For example, in Fig. 4.3b, $N^{q'_1,min}_{r_0} = 1$ since r_0 is true if at least one robot is in region r_0, and $N^{q'_0,min}_{r_0} = 3$ because there should be at least three robots to make proposition T_r_0 become true in state q'_0 (form a triangle). In this work we determine $N^{q',min}_{r_i}$ and $N^{q',max}_{r_i}$ based on the procedure in [16].

We create three sets of linear constraints to ensure the sub-swarm sizes satisfy all the transitions in $\mathbb{A}'$. First, for any transition (q', p'), the sum of outgoing robots from a region r_i must fall in the range $[N_{r_i}^{q',min}, N_{r_i}^{q',max}]$:

$$N_{r_i}^{q',min} \leq \sum_{\mathbf{r}_{p'}} x_{r_i,\mathbf{r}_{p'}}^{q',p'} \leq N_{r_i}^{q',max}, \tag{4.1}$$

where $\mathbf{r}_{p'}$ is the set of neighboring regions of r_i in state p'. Second, for every transition (q', p'), the sum of incoming robots to a region r_i must fall in the range $[N_{r_i}^{p',min}, N_{r_i}^{p',max}]$:

$$N_{r_i}^{p',min} \leq \sum_{\mathbf{r}_{q'}} x_{\mathbf{r}_{q'},r_i}^{q',p'} \leq N_{r_i}^{p',max}, \tag{4.2}$$

where $\mathbf{r}'_q$ is the set of neighboring regions of r_i in state q'. Finally, $\forall q' \in Q'$, $\forall u' \in \mathbf{pre}(q')$, and $\forall v' \in \mathbf{post}(q')$, the sum of robots entering or staying in region r_i during the transition (u', q'), is equal to the sum of robots leaving or staying in region r_i during (q', v').

$$\sum_{\mathbf{r}_{u'}} x_{\mathbf{r}_{u'},r_i}^{u',q'} = \sum_{\mathbf{r}_{v'}} x_{r_i,\mathbf{r}_{v'}}^{q',v'}, \tag{4.3}$$

where $\mathbf{r}_{u'}$ and $\mathbf{r}_{v'}$ are neighboring regions of r_i in states u' and v' respectively. In the end, we formulate two integer programs:

$$min \sum_{(q',p')\in\delta'} x_{\mathbf{r}_{q'},\mathbf{r}_{p'}}^{q',p'} \text{ subject to Eq.}(5.1),(5.2),and(5.3) \tag{4.4}$$

$$max \sum_{(q',p')\in\delta'} x_{\mathbf{r}_{q'},\mathbf{r}_{p'}}^{q',p'} \text{ subject to Eq.}(5.1),(5.2),and(5.3) \tag{4.5}$$

to obtain the upper and lower bounds of number of robots for each pair of neighboring regions in each transition. We use the resulting numbers in the decentralized assignment (section 4.4.2) to make sure that the execution can satisfy all the physical constraints. These constraints might not be the most permissive if we were only considering single transitions (See the example in Sec. 4.5.2).

4.4.2 Auction-based decentralized assignment

Inspired by decentralized auction-based task assignment, we present our swarm assignment algorithm that, based on $\mathbb{A}$ and the robot number constraints (Sec. 4.4.1), assigns robots to different sub-swarms and goals. We execute the assignment algorithm twice for each transition in $\mathbb{A}$, as shown in Fig. 4.2; the first time robots choose their sub-swarm, i.e. their region and formation, and the second time, given the sub-swarm, robots choose goal points to travel to.

Algorithm 2, which builds on [52], describes the distributed assignment algorithm. We consider a set of tasks T; these tasks are region and formation pairs for the first assignment, and goal points to reach for the second. For each task $t \in T$ we assign N_t^{min} and N_t^{max}, the minimum and maximum number of robots that can be assigned to task t. For the sub-swarm assignment, these numbers are given by the IP discussed in Section 4.4.1; for the goal assignments, $N_t^{min} = N_t^{max} = 1$ since we are assigning each robot to exactly one point in the workspace. We assume the communication graph is connected (the same assumption as in [52]) to ensure the convergence of the assignment algorithm.

Notation: We define $T^* \subseteq T$ as the set of tasks the robot can be assigned to. This set is determined before the assignment algorithm is performed, based on the current position of the robot, the physical connectivity of regions, and the next state in the automaton. For example, for the transition between q_1 and q_3 in the automaton in Fig. 4.3a, if the robot was in region r_0 in q_1, $T^* = \{r_0, r_1\}$ due to the environment connectivity (it cannot magically transport to r_2).

During the assignment algorithm, for each $t \in T^*$, we update each robot's estimate of the minimum number of robots still needed for the task, n_t^{min}, and the maximum number

of robots that can still be assigned to the task, n_t^{max} (task availabilities). We denote the set of current minimum and maximum numbers for the set of tasks as $n_{T^*}^{min}$ and $n_{T^*}^{max}$ respectively. Furthermore, we define T^{need} and T^{avail} such that $T^{need} = \{t \in T^* | n_t^{min} > 0\}$ is the set of tasks that must have robots assigned, i.e. the task has not yet been assigned its minimum number of robots, and $T^{avail} = \{t \in T^* | n_t^{max} > 0\}$ is the set of tasks which may still be assigned to robots, i.e. their maximum constraint has not been exceeded. We denote the robot's set of neighbors, the robots within communication range, as $\mathcal{N}$.

Algorithm: At the beginning of each assignment, we initialize $n_{T^*}^{min}$ and $n_{T^*}^{max}$ as noted above. Following the initialization, the assignment is done in bidding rounds (Lines 4-22) until the robot finalizes its task. At each round, the robot first updates the availability of the tasks based on the information it receives from its neighbors $\mathcal{N}$ (Lines 4-5). If it is the first round of bidding or the task t the robot has chosen in a previous round is no longer available, the robot chooses a new task t and an associated bid b as described below (Line 7). Then, the robot determines which of its neighbors are competing for the same task and what their bids are. If there are no neighbors or all their bids are lower, the robot finalizes its task selection and updates the task availability (Lines 14-15). If there is an identical bid, the robot increases its bid (Line 18), and if the robot bid is lower than a competing bid, the robot selects a task again. (Line 21).

Choosing t and b: The function *chooseTask* (Line 7) chooses a task t from T^{need} and T^{avail}. Specifically, in this paper, we selects t from T^{need} if $T^{need} \neq \emptyset$ and from T^{avail} if $T^{need} = \emptyset$ such that the traveling distance from robot i to task t is minimal. We choose the bid b as the reciprocal of the distance to the formation center/goal point. We increase bids by multiplying them by 2 in cases when they are the same (Line 18), but due to our choice of bid, robots almost never have identical bids since the distances are unlikely to

be exactly the same.

Algorithm 2: Distributed Auction for Task Assignment

Input: T^*, N_t^{min}, N_t^{max} $\forall t \in T^*$, $\mathcal{N}$

Output: The robot assignment t

1: $n_t^{min} := N_t^{min}$, $n_t^{max} := N_t^{max}$ $\forall t \in T^*$

2: finalized := False

3: **while** !finalized **do**

4: $(n_{T^*}^{min}, n_{T^*}^{max}) := updateAvailabilities(\mathcal{N})$

5: $(T^{need}, T^{avail}) := updateTaskSet(n_{T^*}^{min}, n_{T^*}^{max}, T^*)$

6: **if** first time bidding or $t \notin T^{avail}$ **then**

7: $t := chooseTask(T^{need}, T^{avail})$, $b := createBid(t)$

8: continue

9: **end if**

10: $(\mathcal{N}_t, B) := findNeighborBids(t, \mathcal{N})$

11: **if** $\mathcal{N}_t = \emptyset$ or ($\mathcal{N}_t \neq \emptyset$ and $b > max_{i \in \mathcal{N}_t}(b_i \in B)$) **then**

12: //No other robots are bidding on t,

13: //or b is the highest bid

14: finalized := True

15: $n_t^{min} := n_t^{min} - 1$, $n_t^{max} := n_t^{max} - 1$

16: **else if** $b_i = max_{i \in \mathcal{N}_t}(b_i \in B)$ **then**

17: // Another robot has the same bid for task t

18: $b := increaseBid(b)$

19: **else**

20: // Another robot has a higher bid for task t

21: $t := chooseTask(T^{need}, T^{avail})$, $b := createBid(t)$

22: **end if**

23: **end while**

Correctness and Completeness: In each bidding round in Algorithm 2, a robot selects a task from T^{need} until all tasks have reached N_t^{min}, and it always selects a task from T^{avail}, thus the auction is guaranteed to satisfy the sub-swarm size constraints. In each auction round, except the first one, at least one robot finalizes their selection and updates the availability, which is then communicated to the other robots. Therefore, such decentralized auction process ends within a bounded time $(n(n+1)/2$ auctions as in [52]).

4.5 Demonstration and Evaluation

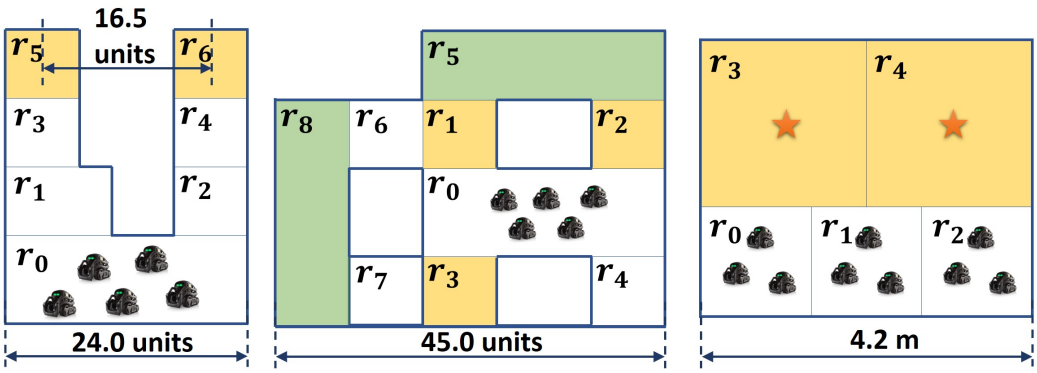


Figure 4.4: Workspace of examples in Sec 4.5. From left to right: small environment, complex task, physical demonstration.

4.5.1 Simulation - small environment

Task description

In the workspace shown in Fig. 4.4 (left), robots are initially located in the charging base r_0. There is an environment signal e such that robots should repeatedly form a square

in r_5 and a hexagon in r_6 when e is true, and stay in r_0 when e is false. We enforce that there should be no robots in r_0 when there are robots in r_3, r_4, r_5, or r_6, and vice versa.

High-level synthesis

We synthesize an automaton $\mathbb{A}$, using the approach in section 4.3.2, which contains 15 states and 30 transitions, and abstract it to $\mathbb{A}'$ with 5 states and 12 transitions. The automaton $\mathbb{A}$ contains transitions where the swarm is split into two sub-swarms, one going to r_5 and the other to r_6 when e is true. Using the approach in section 4.4.1, we calculate the bounds on the number of robots for these two sub-swarms as $[4, 36]$ and $[6, 36]$ respectively.

Evaluation

We evaluate our approach along two dimensions: the effect of choosing a decentralized approach on the average distance traveled by the robots, and the effect of the swarm size on the average number of bids per transition. Fig. 4.5 summarizes the results.

The centralized approach is formulated as an IP that centrally assigns robots to sub-swarms with the objective of minimizing the sum of estimated travel distances towards the target formations [16]. For the evaluation, we vary the swarm size from 10 to 70 (based on the constraint calculation, the swarm size should be between 10 and 72 inclusively), and evaluate average distance traveled by the robots and the number of bids per transition. For each swarm size, we collect data for 20 runs; we randomly sample 20 sets of initial poses, and we use the same initial poses for the centralized and decentralized approaches.

The average travel distance is the accumulated distances of all robots averaged by the number of robots. The number of bids per transition is the accumulated number of auctions from the last robot finalizing the task averaged by the number of transitions for the entire execution of the automaton $\mathbb{A}$. To create the behaviors of the swarm, we set e to be true when all robots are in r_0, and false when robots finish forming a square and a hexagon in r_5 and r_6. We stop the simulation when all robots gather in r_0 again. We set the communication range as 16.0 units in the simulation environment to make sure that two sub-swarms always stay connected.

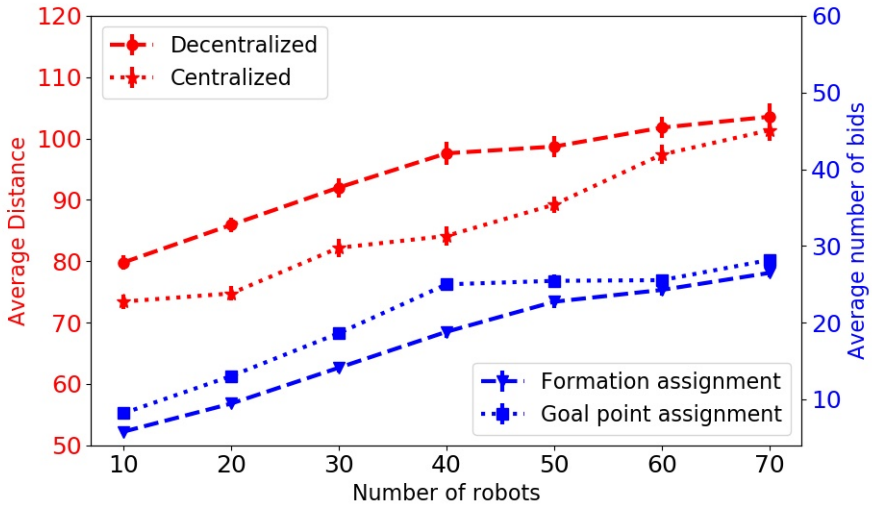


Figure 4.5: Comparison between centralized and decentralized assignment on travel distances (red) and average number of bids needed for robots per transition for the decentralized approach (blue) with respect to the swarm size.

Travel distance: The red lines in Fig. 4.5 present the differences in the average travel distances between the centralized and decentralized approaches. As expected, for all the swarm sizes from 10 to 70, the decentralized approach results in larger average travel distances than the centralized approach, since the decentralized assignment has no guarantees on optimality while the centralized method does. The two approaches have the largest difference in average travel distance when the swarm size is 40. This is due to the asymmetry in the environment; the distance from points in r_0 to r_1 is, on

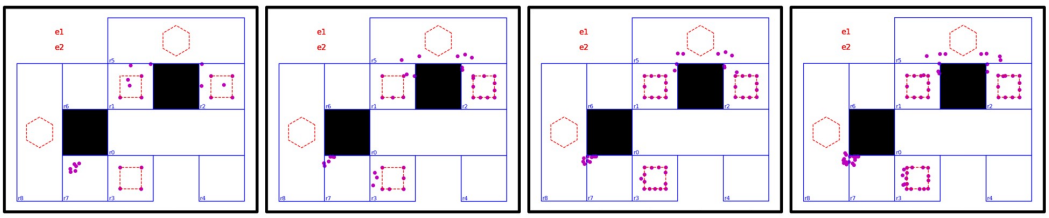


Figure 4.6: Different sizes of simulated swarms performing the same transition (both e_1 and e_2 are true) in $\mathbb{A}$ (Sec. 4.5.2). Swarm sizes are 24, 40, 56, and 72 from left to right. Robots are purple circles, intended formations are displayed in red.

average, smaller than the distance to r_2. This asymmetry causes the optimal (centralized) assignment to assign as many robots as possible to the sub-swarm moving to r_5 , while satisfying $N_{r_2}^{min} = 6$ and $N_{r_1}^{max} = 36$. However, in the decentralized approach, some robots are assigned to r_2 even though they could have been assigned to r_1 with shorter travel distances, because some robots closer to r_1 might first regard r_2 as T^{need}, and after they are assigned to r_2, there are other robots closer to r_2 that choose r_2 from T^{avail}. However, for swarm sizes at the extremes (10 and 70), the difference in travel distance is less noticeable because both approaches will result in almost exactly the same sub-swarm assignment. For a swarm of size 40, the difference is larger because a greater portion of robots are assigned, by the decentralized approach, to r_2.

Average number of bids: The blue lines in Fig. 4.5 present the number of bids needed for robots per transition; each line represents one type of assignment—sub-swarm and goal points. The number of bids grows almost linearly with the number of robots, which shows the scalability of the distributed approach. Therefore, even though the distributed approach cannot guarantee optimality, it is more realistic for applications of large swarms since the computation on each robot scales linearly with the swarm size.

4.5.2 Simulation - complex task

Task description

In the workspace shown in Fig. 4.4 (middle), there are two environment events e_1 and e_2. We require a swarm of robots, initially located in r_0, to form squares in r_1, r_2, and r_3 simultaneously when e_1 is true, and to form hexagons in r_5 and r_8 simultaneously when e_2 is true. We assume that e_1 and e_2 can both be true at the same time, and that once an environment event becomes true, it stays true until the goal is achieved. (E.g., e_1 stays true until the swarm completes forming squares in r_1, r_2, and r_3.)

Synthesis and Demonstration

We synthesize the automaton $\mathbb{A}$ which has 48 states and 125 transitions and abstract it to $\mathbb{A}'$ which has 18 states and 52 transitions. We compute the sub-swarm size during each transition based on $\mathbb{A}'$, which shows that when e_1 is true, the minimum number of robots sent from r_0 to r_3 is 10 (constrained by the minimum number required to form a square in r_3 and a hexagon in r_8), and the minimum number from r_0 to r_1 and r_2 are both seven (constrained by the minimum number required to form squares in r_1 and r_2 and a hexagon in r_5). Similarly, the maximum sub-swarm size from r_0 to r_3 is 36 (constrained by the square in r_3), and the maximum sub-swarm sizes from r_0 to r_1 and r_2 are both 18 (constrained by the hexagon in r_5). From these constraints we know that the swarm should be composed of at least 24 robots and at most 72. If we only allow e_1 or e_2 to be true at a time, the minimum swarm size becomes 14; the maximum swarm size remains 72 due to the maximum number allowed in a square and a hexagon. Fig. 4.6 shows a snapshot of the accompanying video where the swarm distributes into sub-swarms for different swarm sizes (24, 40, 56, and 72) when e_1 and e_2 are both true.

4.5.3 Physical demonstration

Task description

We demonstrate our approach using 10 Anki-Vector robots [1] and a motion capture system. This demonstration simulates a scenario in which robots monitor areas of interest when there is potential danger in the environment. Specifically, robots are initially located in three rooms, r_0, r_1, and r_2 shown in Fig. 4.4 (right). The environment signal d represents danger. Robots are required to go to open spaces r_3 and r_4, and form a square and a hexagon respectively around the targets when d is true, and repeatedly gather and form triangles in the three rooms when d is false. Robots should not be in r_0 and r_4 at the same time.

High-level Synthesis

The synthesized automaton $\mathbb{A}$ contains 18 states and 36 transitions, and $\mathbb{A}'$ has 6 states and 14 transitions. Based on our robot number constraint calculation, We determine that the minimum number of robots required to accomplish the task is 10.

Demonstration

Snapshots of the demonstration are shown in Fig. 4.7. We set the communication range to be $2.8m$ to make sure that the communication graph stays connected when robots are distributed in r_2 and r_3. We control the environmental event "danger" d by clicking a button in a graphical user interface. Initially, d is false and the swarm forms three triangles in r_0, r_1, and r_2. After the completion of the triangles, we change d to be true and the swarm proceeds to form a square and a hexagon around the targets.

We illustrate the bidding process in Fig. 4.7 before each transition in the high-level task. Initially, three robots choose r_3 and seven robots choose r_2 based on the distance to the target regions. Then, before forming the triangles, seven robots bid for staying in the same region at first, but four robots switch the target region to r_1 in order to satisfy the minimum number constraint for each formation. In addition, when moving from $r_3 \wedge r_4$ to $r_2 \wedge r_3$, three robots change the target regions from r_2 to r_3 based on the estimated distances.

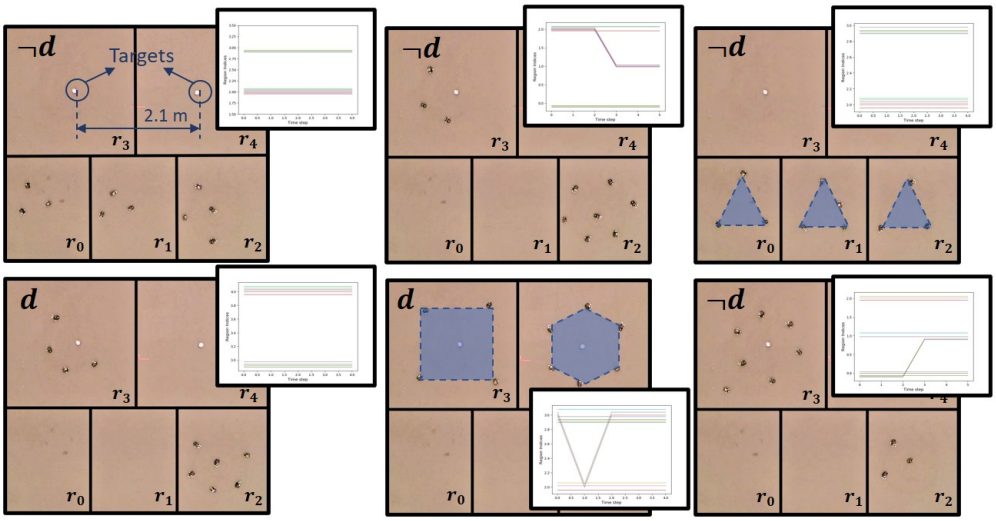


Figure 4.7: Snapshots of 10 Anki Vectors accomplishing a high-level task. The plots represent the region selections of robots over time during the assignment. Each line represents the region selection of one robot over the number of auctions.

4.6 Conclusion ans Discussion

In this work, we develop and demonstrate decentralized execution of high-level reactive swarm tasks. This decentralized approach to executing globally specified tasks retains the formal guarantees of correctness we have shown in the past and enables the algorithms to scale to larger swarm sizes, at the expense of optimality.

Our approach makes several assumptions that may cause limitations in wide scale deployment: First, we assume that all robots bidding for the same task can form a connected communication graph, which might be an obstacle to outdoor applications for real swarms. In our demonstrations, we set a fairly large communication range to satisfy the assumptions, which resulted in a dense communication graph. Thus the bidding process converged quickly. It would be interesting to explore what information needs to be passed and how a restricted communication graph affects the correct execution of the task (see multi-robot exploration tasks [7]).

In addition, the reactivity in this work is non-instantaneous, meaning that the swarm has to finish the current transition before reacting to an environment change. If we allow instantaneous reaction to environment signals, the constraints on sub-swarm size may no longer be satisfied because the environment can choose a behavior where the robots must keep switching sub-swarm, causing livelock. It would be interesting to further explore how to trade-off the instantaneous reaction with the non-instantaneous motion for a specific number of robots.

Finally, when implementing the algorithm on swarm robots, we observe that the robots closer to goal points have higher priority to make the decisions, and they usually reach the goals quickly while other robots have to go around them to reach the goals, which results in two disadvantages: 1) swarms might get congested and have deadlocks in a narrow workspace, and 2) it might take a long time to complete the tasks. Although our approach is decentralized, we can still improve it by incorporating ideas of swapping goals [87] to speed up the execution and avoid deadlocks since each robot plays a cooperative role in the swarm to achieve the task.

CHAPTER 5

HIGH-LEVEL TASKS FOR SWARMS: AUTOMATIC, CORRECT REDISTRIBUTION OF ROBOTS TO SATISFY ON-THE-FLY USER REQUESTS

5.1 Introduction

Swarm robotics studies the robust, scalable, and flexible collective behaviors of large numbers of simple robots with different interaction rules [13]. While traditionally research in swarm robotics has considered bottom-up approaches where researchers designed rules for individual robots and then analyzed the emergent behavior [13], recently there has been interest in developing top-down control schemes [42, 34, 26, 54, 19] which make use of formal languages to capture and generate desired behaviors. In all these approaches, the task specification is given before swarm execution; while these techniques provide guarantees for overall swarm behavior, they cannot adjust, during execution, to a change in the specification. In this paper, we propose an approach that allows users to redistribute robots *during* execution of high-level tasks, given in linear temporal logic, while guaranteeing satisfaction of the task specifications, if possible.

As a motivating example, presented in Fig. 5.1, we consider a swarm of robots executing a high-level task. The swarm may move in the environment, may split into sub-swarms and recombine to larger sub-swarms. We assume the swarm is responsive to an environmental event e which is abstracted as a Boolean variable. The specification for the swarm is:

- Repeatedly visit regions r_5 and r_6 when e is True.
- Repeatedly visit region r_0 when e is False.

For the physical space, there is an upper bound of robot numbers for every region. For example, at most three robots are allowed in r_1 at the same time, while ten robots in all the other regions.

In prior work, we have shown how one can automatically synthesize a symbolic plan and individual robot controls to satisfy such a task [54, 18]. For this example, the approach divides the swarm into two sub-swarms of two and six robots moving towards r_5 and r_6 respectively when e is true, while making all the robots gather in r_0 when e is false. Now, in this work, we consider that during task execution, a user requests a change to the robot numbers in the different sub-swarms, and the swarm should react to the user's request accordingly. For example, in Fig. 5.1, when two sub-swarms are in r_3 and r_4 with e being True, the user requests that the sub-swarm going to r_5 contain five robots instead of two. How should the swarm react to this request and achieve the desired redistribution?

There are two main challenges for accommodating the user's request while maintaining task correctness: i) choosing which robots to move and planning their paths, and ii) ensuring that the original high-level task is not violated during the redistribution. In this paper, we propose a framework that iteratively applies reactive synthesis [12] and integer programming to create a set of tentative symbolic plans, which represent the paths to target regions, and then generates a synchronization skeleton on those tentative plans, which captures when different sub-swarms must synchronize, to ensure that the collective behaviors satisfy both the original specifications and new requests. We demonstrate our approach on simulated swarms and present the computation time of generating synchronization skeletons with respect to region sizes, sub-swarm numbers, transition lengths, and safety specification numbers.

Contribution. In this work we consider high-level swarm tasks and provide a frame-

work that responds to user requests for robot redistribution given *during execution*. In the framework, we use a method to create tentative symbolic plans, which can be executed by sub-swarms, to satisfy robot number constraints in regions. Then, we develop an algorithm to create synchronization skeletons on the symbolic plans to ensure task specifications are satisfied.

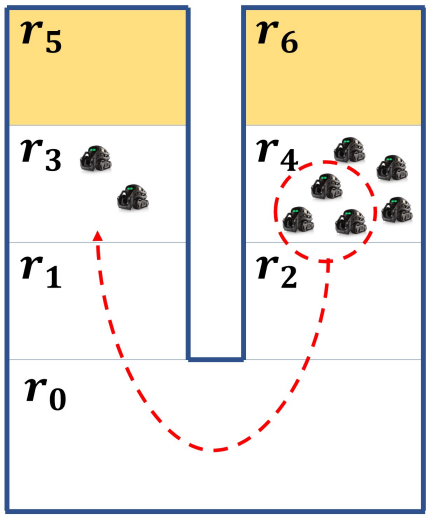


Figure 5.1: A swarm is executing a high-level task, where it is divided into two sub-swarms moving towards r_5 and r_6. During the task execution, a user requests three robots from the sub-swarm on the right to the left. The region r_1 only allows at most three robots at a time.

5.2 Related work

Reactive synthesis for robot swarms. Reactive synthesis [71], and synthesis from tasks specified in different temporal logics, has been applied to robotic systems (see references in [45]). These systems include mobile robots, manipulators, multi-robots systems and swarms. In the context of swarms, [42] presents a hierarchical framework for swarm robots to achieve high-level specifications, where individual velocity commands are synthesized with limited inter-robot communications. Authors in [56]

develop a reactive synthesis approach to solve the swarm navigation problem with guarantees of satisfying linear temporal logic specifications. In [19], we propose a control synthesis framework for non-reactive swarm formation tasks, and further in [18], we develop a decentralized assigning algorithm for swarms to achieve reactive tasks, where the swarm behavior depends on abstracted, uncontrolled environment events. In this paper, we extend the swarm reactivity of [18], allowing the swarms to be responsive to a user's requests regarding robot numbers during the task execution, and generate robot control satisfying all specifications, if possible.

Online path planning for swarm robots. There are many approaches to swarm path planning in the literature. A widely-used approach is particle swarm optimization, where the planning problem is translated into a multi-objective constrained program [47, 24, 36]. The optimal paths are synthesized by formulating the constraints, which consist of spacial and velocity boundaries of each robot, and the objective, which is normally considered as the distance between the robot and its goal. Another way of solving the path planning problem is through search [44, 33], where [44] constructs a probabilistic framework and searches for paths over a Petri Net to satisfy complex specifications, and [33] develops a searching algorithm to achieve optimal task assignment for non-cooperative swarm tasks. In this work, we utilize reactive synthesis [12] to obtain tentative plans by automatically generating linear temporal logic specifications, and applying integer programming and algorithms for constructing synchronization skeletons to those tentative plans to satisfy high-level specifications as well as physical constraints.

5.3 Preliminaries

5.3.1 Integer programming (IP) for swarm size

Using the LTL that we introduce in section 3.3.1, we can use propositions to represent robots in a region. For example, r_i indicates that there are robots currently in r_i, while $\neg r_i$ means no robot is in r_i. By using the conjunction of all the propositions, we can represent the "state" q of the swarm. For example, $r_3 \wedge r_4 \wedge \bigwedge_{i \neq 3,4} \neg r_i$ (or simplified as $r_3 \wedge r_4$) indicates that there are two sub-swarms in regions r_3 and r_4 in Fig. 5.1. Also, the workspace can be represented symbolicly as a region graph $G_r = (\mathbf{R}, \mathbf{E})$, where $\mathbf{R}$ is the set of all regions, and $\mathbf{E}$ is the set of all pairs of neighboring regions which connects in the phyiscal space. A symbolic plan is a a list of transitions that the swarm makes to go from one state to another state, where two connecting (neighboring) states should satisfy the robot motion in the physical space, i.e., sub-swarms only move to neighboring regions from one state to its neighboring state. For a transition (q, p), we call q as the previous state of p, and p as the post state of q.

In [18], we present an IP-based algorithm to determine, before the task execution, the swarm size given a synthesized symbolic plan for the swarm behavior, and physical restrictions on the number of robots allowed in different parts of the environment. We use $N_{r_i}^{q,min}$ and $N_{r_i}^{q,max}$ to represent the minimum and maximum number of robots required in region r_i when the swarm is in state q respectively.

We use $x_{r_i,r_j}^{q,p}$ to represent the number of robots moving from r_i to r_j in the transition (q, p). There are three types of linear constraints. First, for any transition (q, p) in the symbolic plan where q and p are neighboring states, the sum of outgoing robots from a

region r_i must be in the range $[N_{r_i}^{q,min}, N_{r_i}^{q,max}]$:

$$N_{r_i}^{q,min} \leq \sum_{\mathbf{r}_p} x_{r_i,\mathbf{r}_p}^{q,p} \leq N_{r_i}^{q,max}, \tag{5.1}$$

where $\mathbf{r}_p$ is the set of neighboring regions of r_i in state p that have robots. Then, for any transition (q,p), the sum of incoming robots to a region r_i must fall in the range $[N_{r_i}^{p,min}, N_{r_i}^{p,max}]$:

$$N_{r_i}^{p,min} \leq \sum_{\mathbf{r}_q} x_{\mathbf{r}_q,r_i}^{q,p} \leq N_{r_i}^{p,max}, \tag{5.2}$$

where $\mathbf{r}_q$ is the set of neighboring regions of r_i in state q that have robots. Finally, $\forall u \in \mathbf{pre}(q)$, and $\forall v \in \mathbf{post}(q)$, where $\mathbf{pre}(q)$ and $\mathbf{post}(q)$ are previous and post states of q, the sum of robots entering or staying in region r_i during the transition (u,q), is equal to the sum of robots leaving or staying in region r_i during (q,v):

$$\sum_{\mathbf{r}_u} x_{\mathbf{r}_u,r_i}^{u,q} = \sum_{\mathbf{r}_v} x_{r_i,\mathbf{r}_v}^{q,v}, \tag{5.3}$$

where $\mathbf{r}_u$ and $\mathbf{r}_v$ are neighboring regions of r_i in states u and v respectively. Considering all these three constraints in the IP, we can obtain the lower and upper bounds of region-wise robot numbers for each transition before the execution, which can be later used as the constraints when assigning robots to regions.

5.3.2 Synchronization skeletons of sub-swarms to satisfy specifications

To ensure correct execution of high-level swarm tasks, with safety constraints and goals, [58] presents a method for creating a synchronization skeleton, essentially constraints on when robots in the swarm must synchronize, such that the overall behavior of the decentralized system is correct. For each sub-swarm, we have a symbolic plan P_i which represents the sequence of regions it should visit, and limits on robot numbers $\mathcal{N}_i$, that

constrain the number of robots in the sub-swarm, which is the lower bound of $N_{r_j}^{q,max}$ where q is any state in P_i and r_j is the location of the sub-swarm in q. Then, given a set of decentralized symbolic plans P_i, where $P_i = q_0 q_1 ... q_l$, a synchronization skeleton is defined as $\mathbb{F} = f_0 f_1 ... f_k$ where $0 \leq f_i \leq l$ represents sub-swarms should synchronize at state q_{f_i}.

5.4 Problem formulation

We consider the following problem: Given a region graph G_r, n robots, the number of robots distributed in m regions $\mathbf{N}_0 = [N_0^1, ..., N_0^m]$ where $n = \sum_{i=1}^{m} N_0^i$, high-level specifications $\varphi = \varphi_i \wedge \varphi_s \wedge \varphi_g$ where φ_i, φ_s, and φ_g are initial condition, safety and liveness, the maximum robot numbers allowed in regions $\mathbf{N}_{max} = [N_{max}^1, ..., N_{max}^m]$, a user-defined request regarding the number of robots in regions $\mathbf{N}_g = [N_g^1, ..., N_g^m]$, we synthesize a motion plan $\mathbf{P}$ that re-distributes the robots to satisfy the user requests while satisfying φ and $\mathbf{N}_{max}$, or provide feedback to the user if correct re-distribution is not feasible.

5.5 Approach

We describe our approach in Fig. 5.2. First, we synthesize the automaton before task execution using the method in [18]. Then, during the execution, a user gives requests on sub-swarm sizes, and we iteratively synthesize temporal symbolic plans by converting number constraints to LTL specifications. For each symbolic plan that we have synthesized, we check whether the sub-swarm sizes satisfy the robot number constraints for each region: if satisfied, we execute the synthesized plans, and if not, we take the remaining robot numbers and synthesize again. In the end, we execute symbolic plans for

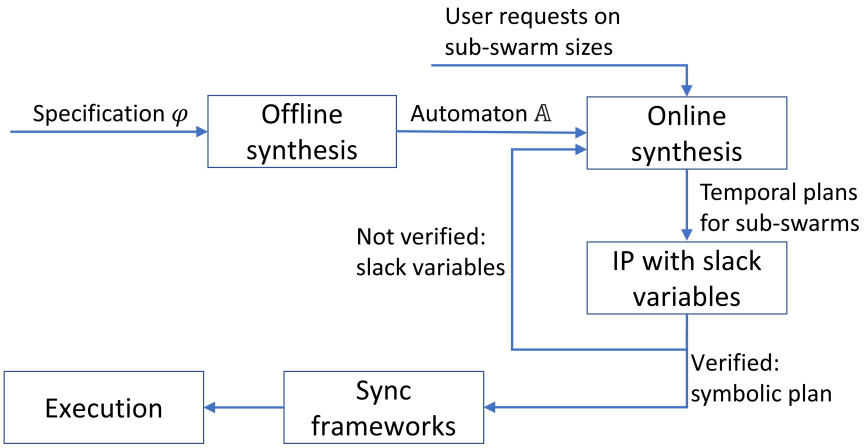


Figure 5.2: Flowchart of our control framework for high-level swarm tasks reactive to user requests.

different sub-swarms and create a synchronization skeleton to guarantee the emergent behaviors satisfy the specifications by leveraging the method in [54].

5.5.1 LTL Specification Generation

We regard the temporal swarm planning as the synthesis problem and generate the LTL specifications automatically. We divide the LTL specifications into 3 parts: region graph, robot number safety, and objectives. For the region graph, we encode how the robots are allowed to move between regions given the physical connectivity in the workspace. Specifically, for every region r_i and all its neighboring regions R_{r_i}, we encode $r_i \rightarrow \bigcirc r_i \vee \bigvee_{r_j \in R_{r_i}} \bigcirc r_j$ which means a robot must stay in the same region or only move to the neighboring region, and $\neg r_i \wedge \bigwedge_{r_j \in R_{r_i}} \neg r_j \rightarrow \bigcirc \neg r_i$ which indicates that a robot cannot enter a region from a non-neighboring region. In addition, we add a safety condition $\neg r_i$ in every iteration if robot number has reached the upper bound from the results in section 5.5.2. For the objectives, we first define $\mathbf{N}_f = [N_f^1, ..., N_f^m]$ as the number flow, where $N_f^i = N_g^i - N_0^i$. We note that a positive N_f^i means region r_i needs more robots and

a negative N_f^i means r_i needs to send out robots. Then, we create initial condition as $\bigwedge r_i$ where r_i are the regions with negative N_f^i, and liveness condition as $\square\Diamond(\bigwedge r_j)$ where r_j has positive N_f^i. Given the specifications consisting of the above three components, we use slugs [27] to synthesize a temporal symbolic plan. Note that we consider the environment proposition remains the same as we execute the transition reacting the user request, and the temporal symbolic plan that we synthesize here is simply a trace (a list of transitions).

5.5.2 Integer Programming with Slack Variables

Given the symbolic plan P_i (a sequence of states q) that we get from section 5.5.1, the transition of any two consecutive states $q_k, q_{k+1} \in P_i$ is the tuple (q_k, q_{k+1}). Then, similar to section 5.3.1 we define a set of non-negative integer variables $x^{q_k,q_{k+1}}_{\mathbf{r}_{q_k},\mathbf{r}_{q_{k+1}}}$, where $\mathbf{r}_{q_k}$ and $\mathbf{r}_{q_{k+1}}$ represent the regions that are in the labels of state q_k and q_{k+1} respectively, such that they are adjacent in the workspace. The variable $x^{q_k,q_{k+1}}_{r_i,r_j} \in x^{q_k,q_{k+1}}_{\mathbf{r}_{q_k},\mathbf{r}_{q_{k+1}}}$ means the number of robots that we actually need to move from r_i to r_j during the transition, to satisfy the robot number constraints, where $r_i \in \mathbf{r}_{q_k}$ and $r_j \in \mathbf{r}_{q_{k+1}}$ are neighboring regions. Different from [18], we introduce non-negative slack variables to this IP to allow soft constraints. We define $s^{q_k,q_{k+1}}_{\mathbf{r}_{q_k}}$ as the slack variables for outgoing constraints of region $\mathbf{r}_{q_k}$, and $u^{q_k,q_{k+1}}_{\mathbf{r}_{q_{k+1}}}$ as the slack variables for incoming constraints of region $\mathbf{r}_{q_{k+1}}$. A positive slack variable indicates how many robots the swarm exceeds the corresponding constraint, which means we need to plan another path for that amount of robots. Note that the number of robots satisfy all the constraints with the given symbolic plan P_i if all slack variables are 0. At this point, we create three sets of linear constraints: outgoing, incoming, and conservation. For the outgoing constraints, the sum of robots from a

region r should satisfy:

$$\sum_{\mathbf{r}_{q_{k+1}}} x_{r,\mathbf{r}_{q_{k+1}}}^{q_k,q_{k+1}} \leq N_r^{q_k,max} + s_{\mathbf{r}_{q_k}}^{q_k,q_{k+1}}, \tag{5.4}$$

which means the number of robots moving out from region r should be less than the maximum number of robots allowed in r plus a buffer with the amount $s_{\mathbf{r}_{q_k}}^{q_k,q_{k+1}}$, and

$$\sum_{\mathbf{r}_{q_{k+1}}} x_{r,\mathbf{r}_{q_{k+1}}}^{q_k,q_{k+1}} \geq 1, \tag{5.5}$$

where $\sum_{\mathbf{r}_{q_{k+1}}} x_{r,\mathbf{r}_{q_{k+1}}}^{q_k,q_{k+1}}$ is the sum of robots moving out from region r to all possible neighboring regions during the transition (q_k, q_{k+1}). For the incoming constraints, the sum of robots moving into a region r should satisfy:

$$\sum_{\mathbf{r}_{q_k}} x_{r,\mathbf{r}_{q_k}}^{q_k,q_{k+1}} \leq N_r^{q_{k+1},max} + u_{\mathbf{r}_{q_{k+1}}}^{q_k,q_{k+1}}, \tag{5.6}$$

which means the number of robots moving into the region r should be less than the maximum number of robots allowed in r plus a buffer with the amount $u_{\mathbf{r}_{q_{k+1}}}^{q_k,q_{k+1}}$, and

$$\sum_{\mathbf{r}_{q_k}} x_{r,\mathbf{r}_{q_k}}^{q_k,q_{k+1}} \geq 1, \tag{5.7}$$

where $\sum_{\mathbf{r}_{q_k}} x_{r,\mathbf{r}_{q_k}}^{q_k,q_{k+1}}$ is the sum of robots moving into r from all neighboring regions during the transition (q_k, q_{k+1}). For the conservation constraints, the defined variables should satisfy for every transition:

$$\sum_{\mathbf{r}_{q_{k-1}}} x_{r,\mathbf{r}_{q_k}}^{q_{k-1},q_k} = \sum_{\mathbf{r}_{q_{k+1}}} x_{r,\mathbf{r}_{q_k}}^{q_k,q_{k+1}}, \tag{5.8}$$

which means the number of robots moving into r during the transition (q_{k-1}, q_k) show be equal to the number of robots moving out from r during the transition (q_k, q_{k+1}). Then, we formulate the IP as:

$$min \sum_{\mathbf{r}_{q_k},\mathbf{r}_{q_{k+1}}} x_{\mathbf{r}_{q_k},\mathbf{r}_{q_{k+1}}}^{q_k,q_{k+1}} + C_1 \sum_{\mathbf{r}_{q_k}} s_{\mathbf{r}_{q_k}}^{q_k,q_{k+1}} + C_2 \sum_{\mathbf{r}_{q_k}} u_{\mathbf{r}_{q_k}}^{q_k,q_{k+1}}, \tag{5.9}$$

where C_1 and C_2 are two large enough constants, since our goal is to minimize the number of robots in each transition and check if they can satisfy the robot number constraints.

Once we solve the IP (5.9), if all the slack variables are 0, we the slack variables $u_{\mathbf{r}_{q_k}}^{q_k,q_{k+1}}$ indicate the robots that should not move into the regions $\mathbf{r}_{q_k}$, which we use as the robot numbers to create another symbolic plans.

5.5.3 Iteration to Obtain the Correct Plan

Using the method in section 5.5.2, we get the maximum sub-swarm size $\mathscr{N}_i$ that can be applied to the synthesized symbolic plan in section 5.5.1. If $\mathscr{N}_i = \sum_{\mathbf{r}_{q_0},\mathbf{r}_{q_1}} x_{\mathbf{r}_{q_0},\mathbf{r}_{q_1}}^{q_0,q_1}$ (we pick one of the transitions to get the total number of robots) is less than remaining the number of robots that we need to transfer to satisfy the user's request, we use the tentative symbolic plans to create synchronization framework and execute the plan. Otherwise, we pick the region r_i that matches the upper bound of the sub-swarm size, and add a safety specification $\square\neg r_i$ to the specifications in section 5.5.1, and rerun synthesis to obtain another symbolic plan. We iteratively execute the IP and safety addition until no more robots need to be transferred. Such process creates a set of symbolic plans associated with robot numbers, which satisfies the sub-swarm size limits.

5.5.4 Obtain optimal plans for sub-swarms

We iteratively generate temporal symbolic plans using the method in section 5.5.1, and in each iteration, we use the approach described in section 5.5.2 to evaluate whether the remaining number of robots can satisfy the constraints.

Sub-swarm redistribution. The result from previous steps is a set of unique symbolic plans, $P_{set} = \{P_1, ..., P_n\}$, where P_i is a symbolic plan consist of a finite sequence of states $q_0^i q_1^i q_2^i ...$ We assume a constant cost to execute a single transition from one state to its neighboring states. The overall cost to complete the redistribution task depends on the plan with the largest number of states in P_{set}. Algorithm 1 describes the redistribution process to minimize the cost for the entire group to complete the task, where the process is to assign robots to different symbolic plans. We iteratively assign robots to plan with the least cost in P_{set} while updating the selected plan by adding two initial states to itself, e.g. $P_i = q_0^i q_1^i q_2^i ... \rightarrow \mathbf{q_0^i q_0^i} q_0^i q_1^i q_2^i ...$, So that we could assign sub-swarms to the "same" plan but execute in series. Adding two initial states instead of one reliefs the issue with the intermediate states during execution, i.e., two sub-swarms running the same route in series could violate the number constraints in a region when robots from both sub-swarms present in the same region. During each assignment, we determine the size of the sub-swarm to be the least of the number of unassigned robots and the maximum sub-swarm size $\mathscr{N}_i$ of the symbolic plan.

Safety guarantee of symbolic plans. We first enforce a feasible fully-synchronous skeleton, where each plan synchronizes with others at every state, then reduce the synchronization steps to obtain the final skeleton, compared to [54] which first obtains asynchronized symbolic plan and then enforces a synchronization skeleton. We adjust the redistribution plans by extending the plan with their last states so that every plan has same number of steps. We enforce the fully-synchronous skeleton by finding synchronization states at every step, e.g. $S_j = \left\{q_j^1, q_j^2, ...q_j^k\right\}$ is the set of synchronous states at step j. Given the safety specification φ_s, a set of states S satisfies φ_s, when φ_s evaluate to be True over all the states in the set. We define a set of plans $\mathbb{P} = \{P_1, ...P_k\} \models \varphi_s$, when all synchronous states in the plans satisfy φ_s. To avoid the issue of intermediate states during execution that may violate the safety specification, we iteratively examine

Algorithm 3: Sub-swarm redistribution symbolic plans

Require: $P_{set} = \{P_1, ..., P_n\}$, where $P_i = q_1^i q_2^i ... q_m^i$, is a symbolic plan.
$\mathscr{N}^{lim} = \{\mathscr{N}_1^{lim}, ..., \mathscr{N}_n^{lim}\}$, where $\mathscr{N}_i^{lim}$ is the maximum number of robots that can be assigned to P_i.
N, number of unassigned robots

Ensure: $\mathbb{P}$, a set of assigned symbolic plans.
$\mathbb{N}$, number of robots assigned to each plan.

1: **while** $N > 0$ **do**
2: find P_i in P_{set} with minimum length
3: $\mathbb{P} \Leftarrow \mathbb{P} \cup P_i$
4: **if** $N >= N_i^{lim}$ **then**
5: $\mathbb{N} \Leftarrow \mathbb{N} \cup N_i^{lim}$
6: **else**
7: $\mathbb{N} \Leftarrow \mathbb{N} \cup N$
8: **end if**
9: $N \Leftarrow N - \mathscr{N}_i^{lim}$
// extend the selected plan by repeat initial state twice at the beginning
10: $P_i \Leftarrow \mathbf{q_1^i q_1^i} q_1^i q_2^i ... q_m^i$
11: **end while**

the set of synchronous states from two continuous time steps to determine if the fully-synchronous plans satisfy φ_s; for example, with two symbolic plans $P_1 = q_0^1 q_1^1 q_2^1 ...$, and $P_2 = q_0^2 q_1^2 q_2^2 ...$, the first set of states we check is $\{q_0^1, q_1^1, q_0^2, q_1^2\}$, and so on. If there is a violation, we search for the plan that is causing the problem, and add standby states to it to ensure the overall plans satisfies φ_s, as descried in Algorithm 2.

Reduce synchronization steps. After we obtain the updated plans with fully-synchronous skeleton, we follow the procedures in algorithm 3 to find the necessary synchronization steps. Reachable states are all the possible interleaving states of the plans if not synchronized. We reinforce synchronization steps if removing one step will create reachable states that violate φ_s, and assemble such steps to the synchronization skeleton. To execute the synchronization skeleton, sub-swarms will execute their plans asynchronously until reaching the next synchronization step in the skeleton, then each group must wait for all others to reach the synchronous states to continue moving.

Algorithm 4: Ensure safety of symbolic plans

Require: $\mathbb{P} = \{P_1, ...P_k\}$: symbolic plans for k subswarms, where $P_m = q_0^m q_1^m ... q_n^m$
φ_s: safety specification

Ensure: $\mathbb{P}_s$: synchronized symbolic plans

initialization: $\mathbb{P}_s \Leftarrow \mathbb{P}$
while $\mathbb{P}_s \not\models \varphi_s$ **do**
  for step $t \in [1, n-1]$ **do**
    $S_t \Leftarrow \{q_t^1, q_t^2, ...q_t^k\}$
    $S_{t+1} \Leftarrow \{q_{t+1}^1, q_{t+1}^2, ...q_{t+1}^k\}$
    if $(S_t, S_{t+1}) \not\models \varphi_s$ **then**
      for subswarm index $i \in [1, k]$ **do**
        $S_{temp} \Leftarrow$ remove q_t^i and q_{t+1}^i from (S_t, S_{t+1})
        if $S_{temp} \models \varphi_s$ **then**
        end if
      end for *// insert standby states to the i_{th} subswarm before // step t*
      $P_i \Leftarrow q_1^i ... q_{t-1}^i \mathbf{q_{t-1}^i q_{t-1}^i} q_t^i, ... q_n^i$
      // match the length of other P with P_i
      $P_j \Leftarrow q_1^j q_2^j ... q_n^j \mathbf{q_n^j q_n^j}, \forall j \in [1, k], j \neq i$
    end if
  end for
end while

Algorithm 5: Reduce synchronization steps

Require: $\mathbb{P}_s$, synchronized symbolic plans.

φ_s, safety specification.

Ensure: $\mathbb{F}$, synchronization skeleton.

$t_{prev} \Leftarrow 0$
for step t in $\mathbb{P}_s$ **do**
  for all reachable states from t_{prev} to $t+1$ **do**
    if any combination violates φ_s **then**
      $\mathbb{F} \Leftarrow \mathbb{F} \cup t$
      $t_{prev} \Leftarrow t$
    end if
  end for
end for

5.6 Demonstrations

5.6.1 Simualted demonstration

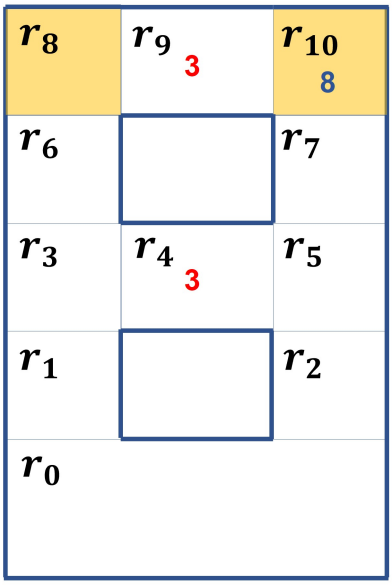


Figure 5.3: The workspace of a simulated example. Robots are required to occupy r_8 and r_{10} simultaneously when e is True, while all robots gather in r_0 when e is False. The regions r_4 and r_9 allows three robots at most, r_0 allows 140, and all the other regions allows 36.

Task description

In the workspace shown in Fig. 5.3, 40 simulated robots are in region r_0 and there is an environmental signal e initially False. The task requires all robots to eventually gather in region r_0 if e is False, and eventually form a square in regions r_8 and a hexagon in r_{10} if e is True. In addition, the task requires that if there are robots in r_6, there should be no robots in r_4 or r_9 and vice versa. We require the regions r_4 and r_9 to have at most three robots at a time for safety. During the task, the robots are divided into two sub-swarms to move towards r_8 and r_{10} respectively. We allow the users to redistribute robots into

the sub-swarms during the execution while not violating the task specifications and the robot number constraints for each region.

Synthesis

As in [18], we synthesize a symbolic plan that makes the robots divide into two sub-swarms and move from r_0 to r_8 and r_{10} when e is true, while gather in r_0 when e is false. The swarm automatically assigns 19 robots to r_8 and 21 robots to r_{10} shown in Fig. 5.4. When the two sub-swarms are entering r_8 and r_{10}, the user requests that 33 and 7 robots should be in r_8 and r_{10} respectively. Such request triggers the process described in section 5.5, where we iteratively synthesize temporal symbolic plans online and check whether the upper limit of robot numbers for each plan satisfies the constraints on regions. In addition, after getting a list of symbolic plans, we use the approach in section 5.5.4 to refine the plans and add synchronization signals to ensure that the sub-swarm behaviors satisfy all the specifications.

Simulation results

The motion of the sub-swrams are shown in Fig. 5.4. After the user sends the command to change robot numbers, the swarm reacts and synthesizes plans after completing the formations in r_8 and r_{10}. First, three robots move towards r_8 through r_9, and two sub-swarms move down to r_5 and split (Fig. 5.4.3). Note that after the first three robots reach r_8, there is another sub-swarm moving from r_{10} to r_8 through r_9 (Fig. 5.4.4), which speeds up the whole process since the route has the least number of region transitions. Then, after the sub-swarm moving through r_4 has entered r_8, another sub-swarm goes through r_9 again, and two robots wait in r_3 until the sub-swarm reaches r_8 (Fig. 5.4.6).

Those sub-swarms synchronize to satisfy the specification that disallows robots in r_6 and r_4 or r_9 at the same time.

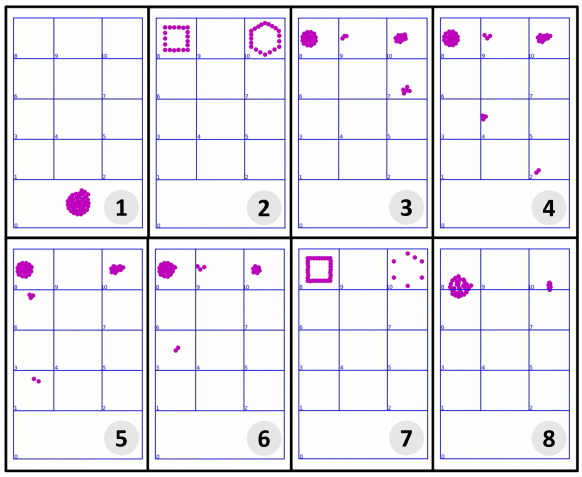


Figure 5.4: Screenshots of the simulated scenario. A user sends a request on robot numbers in (2), and the swarm reacts to the request from (3) to (7). Then, the swarm continues the high-level task in (8).

5.6.2 Comparison between two methods to obtain the synchronization skeletons

Table 5.1: Computation time to create synchronization frames.

R. size	Num.	Trans length	Num. safety	Sync time (Ave/Min/Max)
6	2	5	1	0.00058/0.00012/0.00072
11	3	9	1	0.104/0.1/0.108
11	3	9	2	0.109/0.104/0.113
16	4	13	1	4.237/4.0/4.62
16	4	13	2	4.437/4.01/4.78
16	4	13	3	4.527/4.03/4.84
21	5	17	1	34.235/30.44/37.25
21	5	17	2	34.866/30.07/37.8
21	5	17	3	32.945/29.8/36.5
21	5	17	4	33.637/29.04/37.67

To evaluate the efficiency of our method on the synchronization frame generation,

we collect the computation time with respect to the region sizes, number of sub-swarms, transition length, and the number of safety specifications, where the transition length is represented by the maximum number of states in sub-swarm transitions to complete the user's requirement. We show the results in table 5.1, where the synchronization time is averaged by 10 runs. We note that increasing the number of sub-swarms and transition length will increase the computation time. However, adding the number of safety specifications barely affects the computation, since we evaluate all the safety specifications by creating one conjunction formula.

5.7 Discussion

Automatic control synthesis for robotic swarms is challenging, and making the swarms reactive to human's requests is an important feature in swarm engineering. In this paper, we solve the problem that a synthesis method is applied to create sub-swarm symbolic plans during the task execution to change robot numbers in different regions while satisfying all specifications. Such approach combines the symbolic constraints in the specifications and the physical constraints on robot numbers to synthesize feasible sub-swarm plans, which has the potential in swarm applications where reactivity to human commands are required. However, to further increase the reactivity and optimality of the task execution, we can still make improvements in the future. First, we respond to human's commands at the end of each symbolic transition in this work, which avoiding considering the intermediate states when initializing the temporal sub-swarm plans. In the future, we could check whether starting from the intermediate states will satisfy the specifications, and if so, we can start from the intermediate states to increase the reactivity. In addition, when generating the synchronization skeletons, we first apply fully synchronized plans, and then reducing the synchronized frames to check if the plans

will violate the specifications. Such process can decrease the synchronized numbers, but with no guarantee of optimality. In the future, we could develop a formal method to construct a synchronization skeleton in a bottom-up manner with the least number of synchronized times.

CHAPTER 6

CONCLUSION

The top-down control synthesis approach for swarm robots can adapt to various high-level tasks, such as navigation and shape formation. The control frameworks that I propose provide guarantees on the correctness of the tasks and they remain fully autonomous given the specifications. More importantly, we demonstrate the possibility that the top-down frameworks are also suitable for reactive tasks with in a decentralized manner.

First in Chapter 2, I study the centralized and decentralized control of swarm robotic systems at both the symbolic and continuous levels. At the continuous level, I compare the centralized and decentralized versions of control barrier functions and demonstrate different behaviors when individual failure happens. The results show that decentralized control barrier functions guarantee collision-free motions even with individual failures while centralized computation might cause collisions. At the symbolic level, I use the synthesis framework in [55, 57] to obtain centralized and decentralized symbolic plans, and propose a dynamic assigning method to make swarm robots execute the centralized symbolic plans correctly. I present case studies in simulations and on physical robots to show different behaviors in centralized and decentralized execution. The results show that the execution of the centralized symbolic plan is more efficient in completing the task and more resilient to possible failures as it provides optimal coordination, while the execution of the decentralized symbolic plans is more efficient in computation as all plans are already assigned to robots before the execution.

In Chapter 3, I propose a novel abstraction for robot swarms that allows a user to specify tasks regarding the swarm's formations and locations. I describe a framework for automatically creating control for the swarm robots such that the swarm is guaranteed to

achieve its tasks while avoiding collisions with the environment and between the robots.

In Chapter 4, I develop and demonstrate decentralized execution of high-level reactive swarm tasks. This decentralized approach to executing globally specified tasks retains the formal guarantees of correctness I have shown in the past and enables the algorithms to scale to larger swarm sizes, at the expense of optimality.

In chapter 5, I develop a control framework to automatically redistribute robots to different regions according to users' on-the-fly requests on robot numbers while the swarm is executing high-level tasks. This work incorporate runtime changes to the specifications while maintaining the original high-level specifications in the context of swarms.

BIBLIOGRAPHY

[1] Anki vector, the home robot with interactive ai technology, anki usa. *Available at: https://www.digitaldreamlabs.com/pages/meet-vector*.

[2] Pramod Abichandani, Kyle Levin, and Donald Bucci. Decentralized formation coordination of multiple quadcopters under communication constraints. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3326–3332. IEEE, 2019.

[3] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. Optimal reciprocal collision avoidance for multiple non-holonomic robots. In *Distributed Autonomous Robotic Systems*, pages 203–216. Springer, 2013.

[4] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Roland Siegwart, and Paul Beardsley. Image and animation display with multiple mobile robots. *The International Journal of Robotics Research*, 31(6):753–773, 2012.

[5] Nancy M Amato and Yan Wu. A randomized roadmap method for path and manipulation planning. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 113–120. IEEE, 1996.

[6] Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431. IEEE, 2019.

[7] Francesco Amigoni, Jacopo Banfi, and Nicola Basilico. Multirobot exploration of communication-restricted environments: A survey. *IEEE Intelligent Systems*, 32(6):48–57, 2017.

[8] Jonathan Bachrach, Jacob Beal, and James McLurkin. Composable continuous-space programs for robotic swarms. *Neural Computing and Applications*, 19(6):825–847, 2010.

[9] Jan Carlo Barca and Y. Ahmet Sekercioglu. Swarm robotics reviewed. *Robotica*, 31(3):345–359, 2013.

[10] Calin Belta, Antonio Bicchi, Magnus Egerstedt, Emilio Frazzoli, Eric Klavins, and George J Pappas. Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):61–70, 2007.

[11] Andrew Best, Sahil Narang, and Dinesh Manocha. Real-time reciprocal collision avoidance with elliptical agents. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 298–305. IEEE, 2016.

[12] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Saar. Synthesis of reactive (1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012.

[13] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[14] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, Mar 2013.

[15] Rainer E Burkard and Eranda Cela. Linear assignment problems and extensions. In *Handbook of combinatorial optimization*, pages 75–149. Springer, 1999.

[16] J. Chen, H. Wang, M. Rubenstein, and H. Kress-Gazit. Automatic control synthesis for swarm robots from formation and location-based high-level specifications. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8027–8034, 2020.

[17] Ji Chen, Salar Moarref, and Hadas Kress-Gazit. Verifiable control of robotic swarm from high-level specifications. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 568–576. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[18] Ji Chen, Ruojia Sun, and Hadas Kress-Gazit. Distributed control of robotic swarms from reactive high-level specifications. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 1247–1254. IEEE, 2021.

[19] Ji Chen, Hanlin Wang, Michael Rubenstein, and Hadas Kress-Gazit. Automatic control synthesis for swarm robots from formation and location-based high-level specifications. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8027–8034. IEEE, 2020.

[20] Jianing Chen, Melvin Gauci, Wei Li, Andreas Kolling, and Roderich Groß. Occlusion-based cooperative transport with a swarm of miniature mobile robots. *IEEE Transactions on Robotics*, 31(2):307–321, 2015.

[21] Yang Quan Chen and Zhongmin Wang. Formation control: a review and a new consideration. In *2005 IEEE/RSJ International conference on intelligent robots and systems*, pages 3181–3186. IEEE, 2005.

[22] Han-Lim Choi, Andrew K Whitten, and Jonathan P How. Decentralized task allocation for heterogeneous teams with cooperation constraints. In *Proceedings of the 2010 American Control Conference*, pages 3057–3062. IEEE, 2010.

[23] Aveek K Das, Rafael Fierro, Vijay Kumar, James P Ostrowski, John Spletzer, and Camillo J Taylor. A vision-based formation control framework. *IEEE transactions on robotics and automation*, 18(5):813–825, 2002.

[24] Harshal S Dewang, Prases K Mohanty, and Shubhasri Kundu. A robust path planning for mobile robot using smart particle swarm optimization. *Procedia computer science*, 133:290–297, 2018.

[25] Dimos V Dimarogonas and Karl H Johansson. On the stability of distance-based formation control. In *2008 47th IEEE Conference on Decision and Control*, pages 1200–1205. IEEE, 2008.

[26] Franck Djeumou, Zhe Xu, and Ufuk Topcu. Probabilistic swarm guidance subject to graph temporal logic specifications. In *Robotics: Science and Systems (RSS)*, 2020.

[27] Rüdiger Ehlers and Vasumathi Raman. Slugs: Extensible gr (1) synthesis. In *International Conference on Computer Aided Verification*, pages 333–339. Springer, 2016.

[28] Yousef Emam, Paul Glotfelter, and Magnus Egerstedt. Robust barrier functions for a fully autonomous, remotely accessible swarm-robotics testbed. *arXiv preprint arXiv:1909.02966*, 2019.

[29] E Allen Emerson. Temporal and modal logic. In *Formal Models and Semantics*, pages 995–1072. Elsevier, 1990.

[30] Alessandro Farinelli, Elena Zanotto, Enrico Pagello, et al. Advanced approaches for multi-robot coordination in logistic scenarios. *Robotics and Autonomous Systems*, 90:34–44, 2017.

[31] Fatma Faruq, David Parker, Bruno Laccrda, and Nick Hawes. Simultaneous task allocation and planning under uncertainty. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3559–3564. IEEE, 2018.

[32] John T Feddema, Chris Lewis, and David A Schoenwald. Decentralized control of cooperative robotic vehicles: theory and application. *IEEE Transactions on robotics and automation*, 18(5):852–864, 2002.

[33] Paolo Forte, Anna Mannucci, Henrik Andreasson, and Federico Pecora. Online task assignment and coordination in multi-robot fleets. *IEEE Robotics and Automation Letters*, 6(3):4584–4591, 2021.

[34] Iman Haghighi, Sadra Sadraddini, and Calin Belta. Robotic swarm control from spatio-temporal specifications. *arXiv preprint arXiv:1609.06283*, 2016.

[35] Keliang He, Morteza Lahijanian, Lydia E Kavraki, and Moshe Y Vardi. Reactive synthesis for finite tasks under resource constraints. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5326–5332. IEEE, 2017.

[36] Wenjian He, Xiaogang Qi, and Lifang Liu. A novel hybrid particle swarm optimization for multi-uav cooperate path planning. *Applied Intelligence*, 51(10):7350–7364, 2021.

[37] Gabriel M Hoffmann and Claire J Tomlin. Decentralized cooperative collision avoidance for acceleration constrained vehicles. In *47th IEEE Conference on Decision and Control (CDC)*, pages 4357–4363. IEEE, 2008.

[38] Wolfgang Hönig, TK Satish Kumar, Hang Ma, Sven Koenig, and Nora Ayanian. Formation change for robot groups in occluded environments. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4836–4842. IEEE, 2016.

[39] Jingfu Jin, Yoon-Gu Kim, Sung-Gil Wee, and Nicholas Gans. Decentralized cooperative mean approach to collision avoidance for nonholonomic mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 35–41. IEEE, 2015.

[40] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. Multi-robot task allocation: A review of the state-of-the-art. *Cooperative Robots and Sensor Networks 2015*, pages 31–51, 2015.

[41] Marius Kloetzer and Calin Belta. Hierarchical abstractions for robotic swarms. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 952–957. IEEE, 2006.

[42] Marius Kloetzer and Calin Belta. Temporal logic planning and control of robotic swarms by hierarchical abstractions. *IEEE Transactions on Robotics*, 23(2):320–330, 2007.

[43] Marius Kloetzer, Xu Chu Ding, and Calin Belta. Multi-robot deployment from ltl specifications with reduced communication. In *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pages 4867–4872. IEEE, 2011.

[44] Marius Kloetzer and Cristian Mahulea. A petri net based approach for multi-robot path planning. *Discrete Event Dynamic Systems*, 24(4):417–445, 2014.

[45] Hadas Kress-Gazit, Morteza Lahijanian, and Vasumathi Raman. Synthesis for robots: Guarantees and feedback for robot behavior. *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.

[46] Thomas H Labella, Marco Dorigo, and Jean-Louis Deneubourg. Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(1):4–25, 2006.

[47] Guangsheng Li and Wusheng Chou. Path planning for mobile robot using self-adaptive learning particle swarm optimization. *Science China Information Sciences*, 61(5):1–18, 2018.

[48] Hanjun Li, Chunhan Feng, Henry Ehrhard, Yijun Shen, Bernardo Cobos, Fangbo Zhang, Karthik Elamvazhuthi, Spring Berman, Matt Haberland, and Andrea L Bertozzi. Decentralized stochastic control of robotic swarm density: Theory, simulation, and experiment. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 4341–4347. IEEE, 2017.

[49] Kleber Loayza, Pedro Lucas, and Enrique Peláez. A centralized control of movements using a collision avoidance algorithm for a swarm of autonomous agents. In *Ecuador Technical Chapters Meeting (ETCM), 2017 IEEE*, pages 1–6. IEEE, 2017.

[50] Spyros Maniatopoulos, Philipp Schillinger, Vitchyr Pong, David C Conner, and Hadas Kress-Gazit. Reactive high-level behavior synthesis for an atlas humanoid robot. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4192–4199. IEEE, 2016.

[51] Maja J Matarić, Gaurav S Sukhatme, and Esben H Østergaard. Multi-robot task allocation in uncertain environments. *Autonomous Robots*, 14(2-3):255–263, 2003.

[52] Nathan Michael, Michael M Zavlanos, Vijay Kumar, and George J Pappas. Distributed multi-robot task assignment and formation control. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 128–133. IEEE, 2008.

[53] Dejan Milutinovi and Pedro Lima. Modeling and optimal centralized control of a large-size robotic population. *IEEE Transactions on Robotics*, 22(6):1280–1285, 2006.

[54] Salar Moarref and Hadas Kress-Gazit. Decentralized control of robotic swarms from high-level temporal logic specifications. In *2017 international symposium on multi-robot and multi-agent systems (MRS)*, pages 17–23. IEEE, 2017.

[55] Salar Moarref and Hadas Kress-Gazit. Decentralized control of robotic swarms from high-level temporal logic specifications. In *International Symposium on Multi-Robot and Multi-Agent Systems*. IEEE, 2017.

[56] Salar Moarref and Hadas Kress-Gazit. Reactive synthesis for robotic swarms. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 71–87. Springer, 2018.

[57] Salar Moarref and Hadas Kress-Gazit. Automated synthesis of decentralized controllers for robot swarms from high-level temporal logic specifications. *Autonomous Robots*, pages 1–16, 2019.

[58] Salar Moarref and Hadas Kress-Gazit. Automated synthesis of decentralized controllers for robot swarms from high-level temporal logic specifications. *Autonomous Robots*, 44(3):585–600, 2020.

[59] Petter Nilsson and Necmiye Ozay. Control synthesis for large collections of systems with mode-counting constraints. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 205–214. ACM, 2016.

[60] Kwang-Kyo Oh, Myoung-Chul Park, and Hyo-Sung Ahn. A survey of multi-agent formation control. *Automatica*, 53:424–440, 2015.

[61] Dimitra Panagou, Matthew Turpin, and Vijay Kumar. Decentralized goal assignment and trajectory generation in multi-robot networks: A multiple lyapunov functions approach. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6757–6762. IEEE, 2014.

[62] Christos H Papadimitriou. On the complexity of integer programming. *Journal of the ACM (JACM)*, 28(4):765–768, 1981.

[63] Long Peng, Fei Guan, Luc Perneel, Hasan Fayyad-Kazan, and Martin Timmerman. Decentralized multi-robot formation control with communication delay and asynchronous clock. *Journal of Intelligent & Robotic Systems*, 89(3-4):465–484, 2018.

[64] Kirstin H Petersen, Nils Napp, Robert Stuart-Smith, Daniela Rus, and Mirko Kovac. A review of collective robotic construction. *Science Robotics*, 4(28), 2019.

[65] Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron Ames, Eric Feron, and Magnus Egerstedt. The robotarium: A remotely accessible swarm robotics research testbed. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1699–1706. IEEE, 2017.

[66] Daniel Pickem, Myron Lee, and Magnus Egerstedt. The gritsbot in its natural habitat-a multi-robot testbed. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4062–4067. IEEE, 2015.

[67] Pietro Pierpaoli, Thinh Doan, Justing Romberg, and Magnus Egerstedt. A reinforcement learning framework for sequencing multi-robot behaviors. *arXiv preprint arXiv:1909.05731*, 2019.

[68] Pietro Pierpaoli, Anqi Li, Mohit Srinivasan, Xiaoyi Cai, Samuel Coogan, and Magnus Egerstedt. A sequential composition framework for coordinating multi-robot behaviors. *arXiv preprint arXiv:1907.07718*, 2019.

[69] Giovanni Pini, Arne Brutschy, Mauro Birattari, and Marco Dorigo. Task partitioning in swarms of robots: reducing performance losses due to interference at shared resources. In *Informatics in Control Automation and Robotics*, pages 217–228. Springer, 2011.

[70] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. IEEE, 1977.

[71] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190, 1989.

[72] James A Preiss, Wolfgang Hönig, Nora Ayanian, and Gaurav S Sukhatme. Downwash-aware trajectory planning for large quadrotor teams. In *2017 IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS), pages 250–257. IEEE, 2017.

[73] Amanda Prorok. Robust assignment using redundant robots on transport networks with uncertain travel time. *IEEE Transactions on Automation Science and Engineering*, 17(4):2025–2037, 2020.

[74] Morgan Quigley, Ken Conley, Brian P Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[75] Vasumathi Raman and Hadas Kress-Gazit. Synthesis for multi-robot controllers with interleaved motion. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 4316–4321. IEEE, 2014.

[76] Wei Ren, Randal W Beard, and Ella M Atkins. Information consensus in multivehicle cooperative control. *IEEE Control systems magazine*, 27(2):71–82, 2007.

[77] Indranil Saha, Rattanachai Ramaithitima, Vijay Kumar, George J Pappas, and Sanjit A Seshia. Automated composition of motion primitives for multi-robot systems from safe ltl specifications. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1525–1532. IEEE, 2014.

[78] Martin Saska, Vojtěch Vonásek, Jan Chudoba, Justin Thomas, Giuseppe Loianno, and Vijay Kumar. Swarm distribution and deployment for cooperative surveillance by micro-aerial vehicles. *Journal of Intelligent & Robotic Systems*, 84(1-4):469–492, 2016.

[79] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. Auctioning over probabilistic options for temporal logic-based multi-robot cooperation under uncertainty. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7330–7337. IEEE, 2018.

[80] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. *The international journal of robotics research*, 37(7):818–838, 2018.

[81] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[82] Yang Song and Jason M O'Kane. Forming repeating patterns of mobile robots: A

provably correct decentralized algorithm. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5737–5744. IEEE, 2016.

[83] Onur Soysal and Erol Sahin. Probabilistic aggregation strategies in swarm robotic systems. In *IEEE Proceedings of Swarm Intelligence Symposium (SIS)*, pages 325–332. IEEE, 2005.

[84] Douglas Vail and Manuela Veloso. Multi-robot dynamic role assignment and coordination through shared potential fields. *Multi-robot systems*, pages 87–98, 2003.

[85] Jur Van Den Berg, Jamie Snape, Stephen J Guy, and Dinesh Manocha. Reciprocal collision avoidance with acceleration-velocity obstacles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3475–3482. IEEE, 2011.

[86] Jur P Van Den Berg and Mark H Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, 21(5):885–897, 2005.

[87] Hanlin Wang and Michael Rubenstein. Shape formation in homogeneous swarms using local task swapping. *IEEE Transactions on Robotics*, 36(3):597–612, 2020.

[88] Hanlin Wang and Michael Rubenstein. Walk, stop, count, and swap: Decentralized multi-agent path finding with theoretical guarantees. *IEEE Robotics and Automation Letters*, 5(2):1119–1126, 2020.

[89] Li Wang, Aaron D Ames, and Magnus Egerstedt. Safety barrier certificates for collisions-free multirobot systems. *IEEE Transactions on Robotics*, 2017.

[90] Alan FT Winfield and Julien Nembrini. Safety in numbers: Fault tolerance in robot swarms. *International Journal on Modelling Identification and Control*, 1(ARTICLE):30–37, 2006.

[91] Alan FT Winfield, Jin Sa, Mari-Carmen Fernández-Gago, Clare Dixon, and Michael Fisher. On formal specification of emergent behaviours in swarm robotic systems. *International journal of advanced robotic systems*, 2(4):39, 2005.

[92] Xiangru Xu, Paulo Tabuada, Jessy W Grizzle, and Aaron D Ames. Robustness of control barrier functions for safety critical control. *IFAC-PapersOnLine*, 48(27):54–61, 2015.

[93] Guang Yang, Bee Vang, Zachary Serlin, Calin Belta, and Roberto Tron. Sampling-based motion planning via control barrier functions. In *Proceedings of the 2019*

3rd International Conference on Automation, Control and Robots, pages 22–29, 2019.