Verifiable Control of Robotic Swarm from High-level Specifications

Robotics Track

Ji Chen
Cornell University
Ithaca, New York
jc3246@cornell.edu

Salar Moarref
Cornell University
Ithaca, New York
sm945@cornell.edu

Hadas Kress-Gazit
Cornell University
Ithaca, New York
hadaskg@cornell.edu

ABSTRACT

Designing controllers for safe, scalable and flexible collective behaviors of large numbers of robots is an important and challenging problem in swarm robotics. In this paper, we focus on provably-correct controller synthesis from high-level specifications and demonstrate the approach on several physical swarms. To this end, we first automatically synthesize discrete controllers (symbolic plans) from high-level task specifications expressed in temporal logic. Then, we automatically synthesize continuous controllers that implement the symbolic plans while ensuring collision avoidance and describe methods for mitigating deadlocks that might occur. In addition, centralized and decentralized continuous controller design are compared and analyzed. Finally, we demonstrate the flexibility and versatility of the control paradigm by applying it to three different examples of swarm systems with two different types of robots.

KEYWORDS

Swarm robotics; Formal methods; Controller synthesis

ACM Reference Format:
Ji Chen, Salar Moarref, and Hadas Kress-Gazit. 2018. Verifiable Control of Robotic Swarm from High-level Specifications. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

Swarm robotics research, inspired by self-organized social animals, aims to make a large number of robots have intelligent collective behaviors [7]. There are many advantages for swarm robotic systems, including larger sensing capabilities, better exploring abilities, inherent fault tolerance, as well as cheaper and simpler individual robot design [4]. However, how to control swarm robots to perform global tasks as well as behave safely remains a challenging problem as the number of robots is typically large [24].

Work in swarm robotics often focuses on the bottom-up design of local rules for individual robots that create emergent swarm behaviors; a trial and error process based on iterative design and testing is an essential part of many existing design methods [7]. In contrast, in [18], we took a top-down approach and considered the following problem: how can one specify a desired collective behavior and automatically synthesize decentralized controllers to achieve the collective objective in a provably correct way? A formal specification language for the high-level description of swarm behaviors was proposed on both *swarm* and *individual* levels, and algorithms were developed for automated synthesis of *decentralized* controllers and *synchronization skeletons* that describe how groups of robots must coordinate to satisfy the specification. In this paper, we focus on the continuous implementation of the symbolic plans constructed using the framework introduced in [18] and guarantee the correct and safe swarm behavior at the continuous level.

As a motivating example, consider a workspace divided into 4 regions A, B, C, and D as shown in Fig. 1. Assume the task requires (1) all the robots in the swarm to gather in regions A and B repeatedly, i.e., part of the swarm must occupy region A while the rest of the swarm is in region B and this behavior must happen repeatedly during the execution, (2) all the robots to visit region C repeatedly, however, they can do so at different times, i.e., it is not required that the whole swarm be present in C at the same time, and (3) when part of the swarm is in region C, the rest of the swarm cannot be in regions A or B and vice versa, when robots are in A or B, there should not be any robots in C.

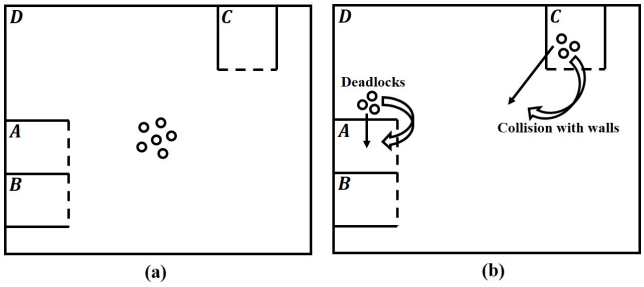

Figure 1: (a) Workspace (b) Possible problems arising during the continuous execution.

Several issues related to correct swarm behavior need to be considered, as illustrated in this example: (1) The collective swarm behavior during the execution must satisfy the specified task, (2) the robots must not collide with each other during the execution, (3) The robots must avoid collision with the obstacles in their environment, and (4) The resulting system should be free of deadlocks, i.e., the robots must always be able to make progress toward their goals.

To address these challenges, we approach swarm control synthesis at two levels: (1) given task specifications, we synthesize discrete controllers that guarantee the swarm robots will complete the task

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

(symbolic plans), and (2) given a symbolic plan, we design continuous controllers that implement the symbolic plans while ensuring collision avoidance as well as deadlock mitigation (continuous).

To validate the proposed swarm control framework, we synthesize controllers for different physical swarm robotic systems - Robotarium [21], a remotely accessible swarm testbed, and Sphero SPRK robots, small rolling sphere robots. We automatically generate discrete (symbolic) controllers from the high-level tasks offline, distribute the symbolic controllers to the robots, create the continuous controllers based on the symbolic ones, and execute the continuous controllers in both centralized and decentralized manners.

Related work. The controller synthesis problem for systems with multiple controllable agents from a high-level temporal logic specification is considered in many recent works (e.g., [13–17, 20, 28, 29]). A common theme to these approaches, similar to this paper, is that they first compute a discrete controller satisfying a specification over a discrete abstraction of the system, and then synthesize or implement continuous controllers guaranteed to fulfill the high-level specification. In this work, we focus on swarms of robots where collisions and deadlocks are more likely due to the number of robots, and we consider synthesis of *decentralized* controllers, both on the symbolic and continuous levels. Moreover, we implement and deploy the proposed framework on two swarm robotic platforms with different number of robots, demonstrating that our approach is agnostic to the number of robots in the swarm.

For safety at the continuous level, there are many approaches to multi-agent collision avoidance (e.g. [5, 10, 12]), most of which consider collision avoidance as the primary goal. In this work, high-level symbolic plans are transformed into continuous controllers that faithfully implement them, in addition to guaranteeing safety. In [27], by integrating control barrier functions (CBF) and quadratic programming (QP), continuous controllers are augmented to ensure collision avoidance. The proposed approach enables robots to move towards their objectives most of the time and execute collision avoidance controllers when they really need to, i.e., when some robots get too close to each other and might collide.

Robots also need to avoid collisions with obstacles for safety. Similar ideas of barrier functions and calculating the safe and optimal control inputs are used in [1, 26, 27]. In this paper, we leverage and extend the work of [27] to handle irregular obstacles represented as polygons.

Deadlock situations where robots cannot make progress towards their goals might happen if the robots only use local information. Most works take a detection-avoidance process to mitigate deadlock, e.g., giving a perturbation input [27] or planning an alternative trajectory [11] after detecting that the robots are in a deadlock situation. However, these approaches cannot guarantee that robots will not get into the same deadlock situation again. In [31], a check-wait approach for avoiding deadlocks is used, i.e., the workspace is discretized and the robot checks whether the next state might cause a deadlock; however, for continuous control, it is impossible to check the next state. In this paper we discuss deadlocks at the continuous level, detect precondition for deadlock, and provide a roadmap-based approach for deadlock mitigation.

Contributions. The contributions of this paper are as follows: (i) Extending the continuous controller design methodology of [27] by using barrier certificates to make robots avoid collision with polygonal obstacles and maintain region boundaries, (ii) Mitigating deadlock by building a road map over the workspace and performing path planning online, and (iii) Demonstrating high-level swarm behaviors using different physical swarm systems to illustrate the versatility of the approach.

The rest of the paper is structured as follows: Synthesis of symbolic controllers is briefly introduced in Section 2. The continuous controller design for collision avoidance and deadlock mitigation is described in Section 3. Task specifications of three examples and the resulting symbolic controllers are given in Section 4. Then, results of those demonstrations are discussed in Section 5 and finally conclusions are made in Section 6.

2 SYNTHESIS OF SYMBOLIC CONTROLLERS

The work in [18] showed how to specify a desired collective behavior in a fragment of Linear Temporal Logic (LTL), and how to automatically synthesize decentralized controllers that can be distributed over robots to achieve the collective objective in a provably-correct way. Decentralized controllers can be viewed as local programs, each executed by a group of robots (i.e. subswarms) of variable size.

Synthesized controllers are executed *asynchronously*, i.e., robots can move with different speeds while executing their symbolic plans. However, to ensure that all the objectives are fulfilled, it may be necessary for the robots to synchronize at some points, e.g., if it is required that the whole swarm must be in a region simultaneously, then the robots in different groups must communicate to determine whether the whole swarm is in the particular region or not. Thus, synthesized controllers are automatically augmented with *synchronization skeletons* that describe how groups of robots must coordinate to satisfy the specifications. In this section, we briefly outline the framework of [18].

Linear Temporal Logic (LTL). We use LTL to specify the global objectives of the swarm system. LTL is a formal language consisting of propositions, Boolean and temporal operators. Let AP be a set of Boolean propositions. The syntax of LTL is defined as follows:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \mathcal{U} \varphi$$

where $\pi \in AP$ is a proposition, $\neg$ is negation, $\vee$ is disjunction, $\bigcirc$ is next and $\mathcal{U}$ 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 $\mathcal{L}(\varphi)$, is the set of infinite words that satisfy φ, i.e., $\mathcal{L}(\varphi) = \{w \in (2^{AP})^{\omega} \mid w \models \varphi\}$. Further details can be found in [8].

In the framework proposed in [18], the user is allowed to specify desired swarm behaviors as a temporal logic specification 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. Roughly speaking, there are two types 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 a *part* of the swarm is currently in region r. Moreover,

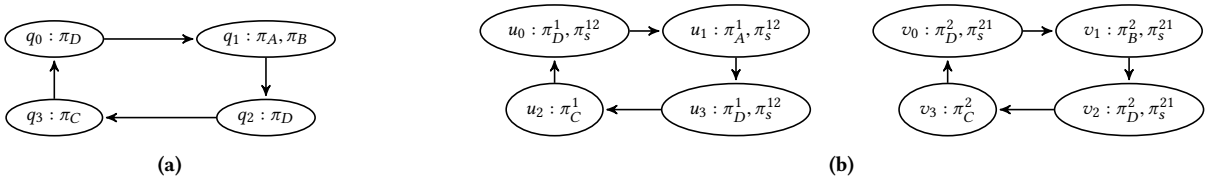

Figure 2: (a) A centralized LTS (b) Partitioning of the centralized LTS of (a) into two LTSs

to distinguish between when we talk about groups or *individual* robots, a parametric proposition is introduced for each region. Let $\pi_r^{\mathbf{a}}$ be a proposition that is true iff an individual robot is in region 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 $\forall \mathbf{a}.\ \Box\Diamond(\pi_r^{\mathbf{a}})$, e.g., $\forall \mathbf{a}.\Box\Diamond(\pi_r^{\mathbf{a}})$ means that all robots must visit region r but not necessarily at the same time. Note that microscopic specification can include safety formulas, e.g., $\forall \mathbf{a}.\Box(\neg\pi_r^{\mathbf{a}})$ (no robot may enter region r). Such safety formulas can also be described at the macroscopic level, e.g., $\Box(\neg\pi_r)$ (no part of the swarm may enter region r). To keep the presentation simple, only liveness formulas are allowed in φ^μ [18]. Furthermore, in this paper, following [18], we assume that the environment is static and known.

Example 2.1. The example outlined in Section 1 can be formally specified with a macroscopic specification $\varphi^M = \Box\Diamond(\pi_A \wedge \pi_B \wedge \neg\pi_C \wedge \neg\pi_D) \wedge \Box(\neg(\pi_C \wedge (\pi_A \vee \pi_B)))$ that indicates (i) the swarm must repeatedly visit regions A and B at the same time and (ii) avoid occupying region C when there are robots in regions A or B and vice versa, and a microscopic specification $\varphi^\mu = \forall \mathbf{a}.\Box\Diamond\pi_C^{\mathbf{a}}$ saying that all robots must repeatedly visit region C, possibly at different times.

Labeled Transition System (LTS). We represent the symbolic controllers as LTSs. An LTS is a tuple $\mathcal{T} = \langle Q, q_0, AP, \delta, \mathcal{L}\rangle$ where Q is a finite set of states, $q_0 \in Q$ is an initial state , AP is a set of propositions, $\delta \subseteq Q \times Q$ is a transition relation, and $\mathcal{L} : Q \to 2^{AP}$ is a labeling function which maps each state to a set of propositions that are true in that state. A *run* of an LTS is an infinite sequence of states $q_0q_1q_2\ldots$ where $q_i \in Q$ and $(q_i, q_{i+1}) \in \delta$, $\forall i \geq 0$. The language of an LTS $\mathcal{T}$ is defined as the set $\mathcal{L}(\mathcal{T}) = \{\mathcal{L}(q_0)\mathcal{L}(q_1)\mathcal{L}(q_2)\cdots \in (2^{AP})^\omega \mid q_0q_1q_2\cdots \text{ is a run of } \mathcal{T}\}$, i.e., the set of (infinite) words generated by the runs of $\mathcal{T}$. An LTS $\mathcal{T}$ *realizes* an LTL specification φ iff all infinite words in its language satisfy φ, i.e., $\forall w \in \mathcal{L}(\mathcal{T}).\ w \models \varphi$. Given an LTL specification, the synthesis problem is to find an LTS that realizes it.

Synthesis of Discrete Controllers. In the framework introduced in [18], the user provides a region graph that represents the connectivity of the workspace. They also provide a temporal logic specification describing the objectives of the system. To synthesize decentralized controllers, first a centralized LTS $\mathcal{T}$ is obtained that satisfies the input specification. The next step is to partition $\mathcal{T}$ to obtain a set of decentralized controllers. Each robot is assigned a decentralized controller, based on its assignment to a subswarm. The synthesis process automatically determines the number of required subswarms and the robot assignment to subswarms can be automated. Subswarms may be of variable size. Note that it is assumed that each subswarm moves (almost) together between the regions, e.g., if a subswarm is moving from region A to B, and some members of it reach B, they wait for other members to enter B before executing their next step.

Finally, a synchronization skeleton for each decentralized controller is constructed that indicates when each subswarm must synchronize with other subswarms to satisfy a liveness or safety guarantee, e.g., to gather in some region. Figure 2a shows a centralized LTS automatically synthesized from the specification in Example 2.1. The swarm, initially in D, moves toward regions A and B. After visiting regions A and B and synchronizing, the robots move toward C through D, and this behavior is repeated indefinitely. To ensure that robots do not violate the safety requirement, they synchronize after moving out of A and B, and also after moving out of C. Figure 2b shows partitioning of the LTS in Fig. 2a into two LTSs, each executed by one group of robots. States where group i synchronizes with group j are labeled with a *synchronization* proposition π_s^{ij} for $i, j \in \{1, 2\}, i \neq j$ as shown in Fig. 2b.

3 CONTINUOUS CONTROLLER DESIGN

The synthesized controllers in Section 2 define the transition of the swarm robots in the symbolic abstraction, i.e., transitions from regions to regions. When implementing the controllers on physical swarm robots, the continuous controllers should be designed to guarantee the following properties: (*i*) *correct behaviors*: the continuous trajectory faithfully implements the runs of the synthesized LTS, (*ii*) *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, (*iii*) *collision avoidance*: robots must avoid collision with each other and obstacles in their environment, (*iv*) *deadlock mitigation*: robots should be able to escape deadlock situations. To this end, we synthesize the continuous controllers as follows: First we create nominal controllers based on the geometry of the workspace and the synthesized symbolic plans. Then we modify the nominal controllers as needed to avoid collisions, maintain region boundaries and mitigate deadlocks.

In this section, we describe the system model, the generation of the nominal controller based on the symbolic plan, a safety-enhanced controller that guarantees collision avoidance and region invariance based on control barrier functions (CBF) [3, 6, 19] and a roadmap based approach to mitigate deadlocks.

3.1 System model

While the approach generalizes to different robot models, in this paper we consider a swarm robotic system with N planar mobile

robots, which are indexed by $\mathcal{M} = \{i \mid i = 1, 2, ..., N\}$. Every single 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 $||\mathbf{v}_i||_\infty \leq \alpha_i$, where α_i is the maximum speed of agent i. For safety in continuous space, we require that any two robots keep a safety distance D_s away from each other at all times. That yields

$$||\Delta \mathbf{p}_{ij}|| \geq D_s \tag{1}$$

where $\Delta \mathbf{p}_{ij} = \mathbf{p}_i - \mathbf{p}_j$ as shown in Fig. 3(a).

3.2 Nominal controller

An intuitive way to design continuous controllers is to use a goal point in the 2D Cartesian space to represent each region, which makes a run of the LTS become a trajectory connecting several goal points. Then, an effective controller for every single robot is a vector pointing to the goal position. We use this approach for defining the nominal controller. In this paper, a proportional controller is used as the nominal controller, which is a vector with the norm proportional to the distance from the current position to the goal position.

3.3 Collision avoidance and region invariance

In [27], the collision avoidance problem for swarm robots was formulated and solved using *control barrier function* (CBF) and *quadratic programming* (QP), where the CBF provides linear constraints on the control input that guarantee that the system stays within a safe set, and QP is done in real-time to modify the nominal controller as little as possible while satisfying the constraints on the control input. In this paper, we leverage the work of [27] and construct CBF that guarantee swarm robots do not collide with each other as before, but also do not collide with polygonal obstacles in the environment and maintain region invariance. In the following, we review the work of [27] (Sections 3.3.1-3.3.3) and describe this paper's contributions (Section 3.3.4-3.3.5)

3.3.1 Control barrier functions (CBF). Consider the general form of a continuous system given as

$$\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x})\mathbf{u} \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state vector of the system, $\mathbf{u} \in \mathbb{R}^m$ is the control input vector, and f and g are locally Lipschitz continuous functions [25]. In this paper, we consider a homogeneous swarm system consisting of N planar robots where $\mathbf{x} = [\mathbf{p}_1^T, \mathbf{p}_2^T, ..., \mathbf{p}_N^T]^T$ is the $2N$ dimensional position vector for all robots, $\mathbf{u} = [\mathbf{u}_1^T, \mathbf{u}_2^T, ..., \mathbf{u}_N^T]^T$ is the $2N$ dimensional control input (velocity vector), $f(\mathbf{x}) = \mathbf{0}$, and $g(\mathbf{x}) = \mathbf{I}_{2N\times 2N}$. Following (1), we define a pairwise safe set ζ_{ij} for every two robots

$$\begin{aligned} &\zeta_{ij} = \{\mathbf{p}_i, \mathbf{p}_j \in \mathbb{R}^2 \mid h_{ij}(\mathbf{p}) \geq 0\} \quad \forall i \neq j, \\ &h_{ij}(\mathbf{p}) = ||\Delta \mathbf{p}_{ij}|| - D_s. \end{aligned} \tag{3}$$

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

$$\zeta = \prod_{i\in M} \{\bigcap_{\substack{j\in M \\ j\neq i}} \zeta_{ij}\} \tag{4}$$

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 [30]. In this paper, 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 (3), the pair-wise CBF can be calculated as

$$B_{ij}(\mathbf{x}, \mathbf{u}) = \frac{\Delta \mathbf{p}_{ij}^T}{||\Delta \mathbf{p}_{ij}||} \Delta \mathbf{u}_{ij} + \gamma h_{ij}^3 \tag{5}$$

where $\Delta \mathbf{u}_{ij} = \mathbf{u}_i - \mathbf{u}_j$. It was proved in [27] that the safe set ζ is forward invariant and asymptotically stable when all $B_{ij}(\mathbf{x}, \mathbf{u})$ are non-negative and the control input $\mathbf{u}$ is Lipschitz continuous. Thus, 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.

3.3.2 Centralized Safety Barrier Certificates. Centralized safety barrier certificates consider all the robots simultaneously, giving control inputs to every robot such that all pair-wise inequality constraints in (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 $b_{ij} = \gamma h_{ij}^3 ||\Delta \mathbf{p}_{ij}||$. Nominal controllers are modified as little as possible to ensure collision avoidance. To this end, QP can be used to minimize the difference between the nominal controller and the actual controller. The QP problem for centralized control is formulated as

$$\begin{aligned} \mathbf{u} = \underset{\mathbf{u}\in\mathbb{R}^{2N}}{\operatorname{argmin}} \quad & J(\mathbf{u}) = \sum_{i=1}^{N} ||\mathbf{u}_i - \hat{\mathbf{u}}_i||^2 \\ s.t. \quad & A_{ij}\mathbf{u} \leq b_{ij}, \quad \forall i \neq j \\ & ||\mathbf{u}_i||_\infty \leq \alpha_i, \quad \forall i \in \mathcal{M}. \end{aligned} \tag{6}$$

Where $\mathbf{u}_i$ is the actual controller and $\hat{\mathbf{u}}_i$ is the nominal controller for agent i, respectively. This way, a $2N$ dimensional continuous control input $\mathbf{u}$ that leads robots to their goals without any collision is computed.

3.3.3 Decentralized Safety Barrier Certificates. The centralized controller requires and computes over full state information of all the robots, which may lead to poor scalability, reactiveness and robustness as the number of robots grows [27]. In order to avoid the drawbacks of centralized control, decentralized safety barrier certificates can be constructed [27]. In decentralized control, the computation is distributed to each robot. To satisfy $B_{ij}(\mathbf{x}, \mathbf{u}) \geq 0$, decentralized safety barrier certificates are formed as

$$-\Delta \mathbf{p}_{ij}^T \mathbf{u}_i \leq \frac{\alpha_i}{\alpha_i + \alpha_j} b_{ij} \tag{7}$$

$$\Delta \mathbf{p}_{ij}^T \mathbf{u}_j \leq \frac{\alpha_j}{\alpha_i + \alpha_j} b_{ij} \tag{8}$$

where i and j represent robot i and robot j, respectively. Note that switching i and j in (7) gives the same constraint as (8). Such decentralized safety barrier certificates distribute the collision avoidance control to each robot according to their agility. Satisfying the safety barrier constraints for all robots will guarantee the collective behaviors are collision-free [27]. In the decentralized case, instead of solving a $2N$-dimensional QP, 2-dimensional QPs in terms of $\mathbf{u}_i$ are solved for each robot i. Detailed formulations for decentralized QP can be found in [27]. The decentralized control approach scales better with the number of robots; however, it might result in deadlocks as it only relies on local information. In Section 4, decentralized barrier certificates will be used in Example 1 and 3, while centralized and decentralized barrier certificates will be compared in Example 2.

3.3.4 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 [27], the authors briefly mentioned that obstacles bounded by circles can be avoided by using control barrier functions and regarding obstacles as virtual robots without control inputs. Here, for first-order systems, we extend the work to collision avoidance with dynamic obstacles which are modeled as virtual robots with limited control inputs.

Let robot i have the smallest velocity bound α_i in the swarm, and dynamic obstacle j have velocity bound β_j. We assume $\beta_j \leq \alpha_i$, otherwise the obstacle can always move toward robot i and cause a collision. Then, we revise the barrier certificate in (7) to be

$$-\Delta\mathbf{p}_{ij}^T\mathbf{u}_i \leq b_{ij} - ||\Delta\mathbf{p}_{ij}||\beta_j \tag{9}$$

where we regard obstacle j as a virtual robot with position $\mathbf{p}_j$. We add all linear constraints (9) to the decentralized barrier certificate to synthesize collision-free continuous controllers for each robot.

Note that when considering two swarm robots, both controllers actively avoid collisions; however in the case of a dynamic obstacle, the obstacle could have an arbitrary velocity within the bound β_j. Eq. (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 (9). In the case of static obstacles, we apply $\beta_j = 0$, which models the static obstacle as a robot with zero control input.

3.3.5 Static polygonal obstacle avoidance and Region Invariance. Section 3.3.4 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, strategies of avoiding collisions with polygonal obstacles are needed for applications of swarm robots in realistic environments. In this section we define safe sets and develop the appropriate barrier certificates 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.

In the following, we explain the control synthesis approach that guarantees the swarm robots avoid collisions with the obstacles.

Theorem 3.1. *For a straight line obstacle, if the controller ensures that the agent is in the safe set ζ_C of the closest point C, then the agent is guaranteed to be in the safe set of the whole straight line ζ_{AB}.*

Proof: Consider a single robot, a line obstacle and a goal position for the robot, as is shown in Fig 3(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 Theorem 3.1, we add constraints by dividing all the polygonal obstacles into lines, finding the closest point to each line, and considering those points as virtual agents with no control inputs. Then, we formulate CBFs in terms of virtual agents as well as real agents and add those constraints to the original QP (6) to compute the control inputs for collision-free motion.

To guarantee region invariance, as described in Section 3, when a subswarm is moving between two regions, we construct virtual obstacles on the boundaries of any other region and follow the procedure above to modify the QP with additional constraints. For example, in Example 2.1, when some robots are required to move from region D to region A, we construct a virtual obstacle at the entrance of region B and use the methods in 3.3.5 to ensure those robots do not enter B.

3.4 Deadlock mitigation

Deadlock is defined as a situation where no progress can be made towards the goal. In swarm motion, when the effects of the barrier certificates counteract the nominal controller, swarm robots might get into a deadlock situation. Consistent perturbations are used in [27] to solve the deadlock problem between the robots themselves, where tangential control inputs in opposite directions are given to two robots. However, the same strategy might not work when dealing with polygonal obstacles. Figure 4(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 because point Q resembles a local minima.

In this section, we address deadlock due to the interaction between a robot and a line obstacle, and then briefly discuss multiple robots and multiple walls scenarios. One-robot-one-wall deadlock happens when the robot is exactly D_s away from the wall and the

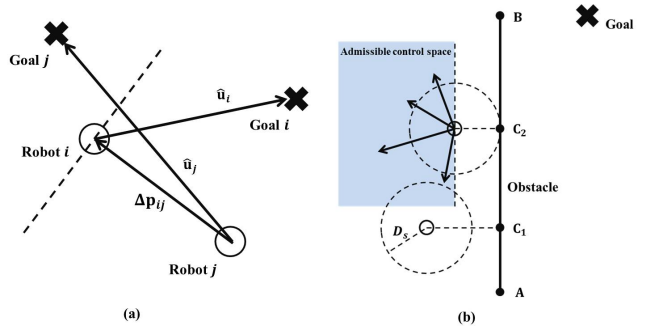

Figure 3: (a) Relative positions between two robots. (b) Relative positions between the robot and the line obstacle.

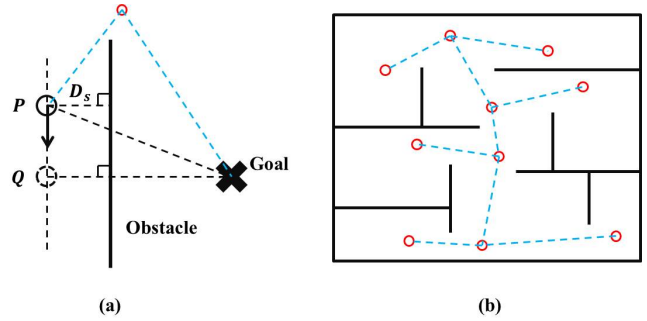

Figure 4: (a) Deadlock scenario when there is a line obstacle between the robot and its goal. (b) Road map over the work space, where red circles represent nodes and blue dashed lines represent edges.

line connecting the robot to the goal perpendicularly intersects the wall (e.g. a robot at point Q in Fig. 4 is in deadlock).

We 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 can 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 environment, we build a roadmap [2] over the workspace, which consists of multiple nodes (waypoints) and edges connecting nodes in free space. As is shown in Fig. 4(b), a roadmap ensures that any starting point can get to any goal point through nodes in a collision-free path. Whenever we detect there is an intersection between the wall and the desired path, we use Dijkstra's algorithm [9] to find another path to the goal point through the connected nodes. Then, the nominal controller is altered to direct to the next waypoint on the roadmap . Note that when applying this deadlock mitigation technique, we ensure that the roadmap does not create paths that violate region invariance. With the roadmap, robots will never get stuck in a deadlock caused by static polygonal obstacles.

In the multi-robot-multi-wall case, deadlocks can be mitigated by tangential perturbation [27] together with the roadmap-based planning, although we cannot guarantee deadlock avoidance in all situations.

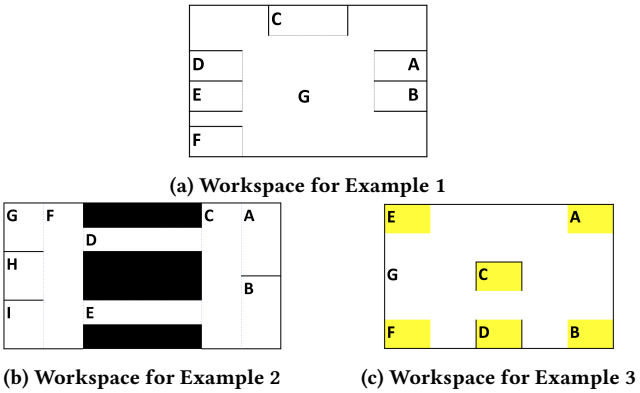

Figure 5: Workspaces for the examples in Section 4

4 PHYSICAL DEMONSTRATION

In this section, we present two different experimental setups and three examples which demonstrate the physical behaviors of swarm robots performing high-level tasks. For examples 1 and 3, the swarm is composed of Sphero SPRK robots. Example 2 was implemented on the Robotarium [21]. Decentralized barrier certificates were implemented for the SPRK robots, while both centralized and decentralized barrier certificates were used and compared in Robotarium.

4.1 Experimental Setup

SPRK robots: SPRK robots[1] are rolling sphere robots with a diameter of 7.3 cm. The robots communicate through Bluetooth and are programmed using the Robot Operating System (ROS) [23]. The control inputs are linear velocities in two perpendicular directions. Thus, the SPRK robots can be modeled as holonomic first-order systems. The Vicon motion capture system was used to localize the robots; we designed rolling cages to enable placing markers on the robots.

Robotarium: The Robotarium system is a remotely accessible swarm testbed developed in [21] where GRITSBots serve as swarm robots. 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 [22].

4.2 Example Description

Example 1. This example shows the ability of the swarm to execute tasks from high-level specifications. We synthesize and deploy controllers on a physical swarm robotic system with 12 robots.

Consider a workspace divided into 7 regions ($A - G$) as shown in Fig. 5a. All robots are initially positioned on the right side of region G. They are required to: repeatedly visit the 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 some 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

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

$\varphi_1^M = \phi_{11} \wedge \phi_{12} \wedge \phi_{13}$. Three LTSs $\mathcal{T}_{11}, \mathcal{T}_{12}, \mathcal{T}_{13}$ with the following runs were synthesized and implemented on three groups of robots:

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

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

Example 2. This example was implemented on the Robotarium platform and shows adaptability of the synthesis framework, i.e., it can be implemented on different physical swarm systems.

Consider two working zones connected by two long corridors shown in Fig. 5b. The swarm, initially distributed in region C, is required to infinitely often navigate through corridors and visit regions G, H, and I simultaneously ($\phi_{21} = \square\diamond(\pi_G \wedge \pi_H \wedge \pi_I \wedge \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{H,G,I\}} \neg\pi_{\mathbf{r}})$). The whole swarm must also repeatedly be in region F ($\phi_{22} = \square\diamond(\pi_F \wedge \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{F\}} \neg\pi_{\mathbf{r}})$, and infinitely often occupy regions A and B at the same time ($\phi_{23} = \square\diamond(\pi_A \wedge \pi_B \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{A,B\}} \neg\pi_{\mathbf{r}})$). Moreover, all the robots in the swarm must pass through corridor E repeatedly ($\varphi_2^\mu = \forall \mathbf{a}.\ \square\diamond(\pi_E^{\mathbf{a}})$) 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 ($\phi_{25} = \square(\neg((\pi_G \vee \pi_H \vee \pi_I) \wedge (\pi_A \vee \pi_B))))$. The macroscopic specification is defined as $\varphi_2^M = \bigwedge_{i=1}^5 \phi_{2i}$. Three LTSs $\mathcal{T}_{21}, \mathcal{T}_{22}, \mathcal{T}_{23}$ with the following runs were synthesized and deployed on three groups of robots:

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

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

Example 3. This example, demonstrated with SPRK robots, shows that two different swarm teams can complete their own tasks safely in a shared workspace regardless of the other team's tasks even though there is no communication between them. This demonstration shows the potential of application in heterogeneous swarms or swarm-human collaboration.

Assume we have two different teams of robots: team 1 with fast robots and team 2 with slow robots. They are required to perform specified tasks in different regions while sharing a public space. We assume that all robots in team 1 are initially positioned in region A, and that they must repeatedly occupy region A at the same time ($\phi_{31}^1 = \square\diamond(\pi_A \wedge \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{A\}} \neg\pi_{\mathbf{r}})$), be present at regions B, E, and F infinitely often at the same time ($\phi_{32}^1 = \square\diamond(\pi_B \wedge \pi_E \wedge \pi_F \wedge \neg\pi_A \wedge \neg\pi_G)$), visit region E repeatedly (possibly at different times) ($\varphi_{31}^\mu = \forall \mathbf{a}.\ \square\diamond(\pi_E^{\mathbf{a}})$), and finally avoid regions C and D at all times ($\phi_{33}^1 = \square(\neg(\pi_C \vee \pi_D))$) since those regions are working areas for team 2. On the other hand, robots in team 2, initially positioned in region C, must repeatedly distribute in region C and D at the same time ($\phi_{31}^2 = \square\diamond(\pi_C \wedge \pi_D)$). Besides, all robots in team 2 must gather in region D repeatedly ($\phi_{32}^2 = \square\diamond(\pi_D \wedge \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{D\}} \neg\pi_{\mathbf{r}})$. Finally, team 2 must avoid entering regions A, B, E and F ($\phi_{33}^2 = \square(\neg(\pi_A \vee \pi_B \vee \pi_E \vee \pi_F))$). The macroscopic specification φ_{3i}^M for team $i \in \{1, 2\}$ is defined as $\varphi_{3,i}^M = \bigwedge_{j=1}^3 \phi_{3j}^i$. Three LTSs $\mathcal{T}_{31}^1, \mathcal{T}_{32}^1, \mathcal{T}_{33}^1$ were synthesized for team 1 with the runs

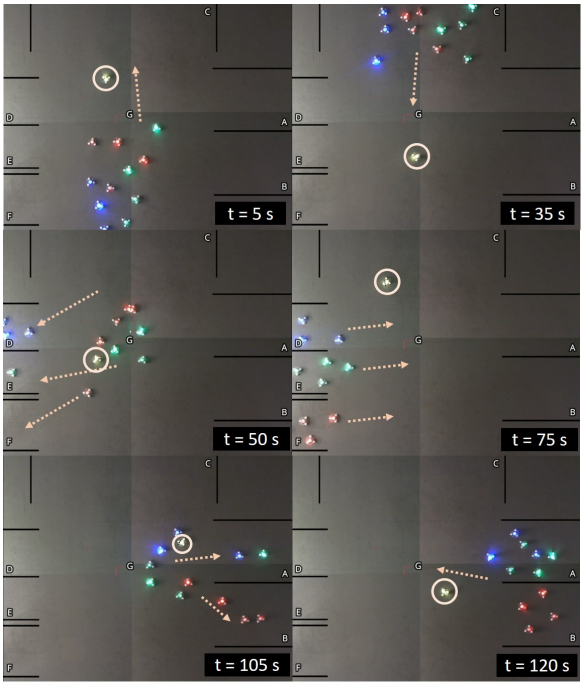

Figure 6: Example 1. SPRKs are divided into three groups specified by green, blue and red. Lines represent walls in the workspace and dashed arrows indicate the motion. The dynamic obstacle is circled.

- $\mathcal{T}_{31}^1$: $\pi_A \rightarrow (\pi_G \rightarrow \pi_E \rightarrow \pi_G \rightarrow \dot{\pi}_A \rightarrow \pi_G \rightarrow \dot{\pi}_B)^\omega$,
- $\mathcal{T}_{32}^1$: $\pi_A \rightarrow (\pi_G \rightarrow \pi_E \rightarrow \pi_G \rightarrow \dot{\pi}_A \rightarrow \pi_G \rightarrow \dot{\pi}_E)^\omega$,
- $\mathcal{T}_{33}^1$: $\pi_A \rightarrow (\pi_G \rightarrow \pi_E \rightarrow \pi_G \rightarrow \dot{\pi}_A \rightarrow \pi_G \rightarrow \dot{\pi}_F)^\omega$.

For team 2, two LTSs $\mathcal{T}_{31}^2, \mathcal{T}_{32}^2$ were synthesized with the runs:

- $\mathcal{T}_{31}^2$: $\pi_C \rightarrow (\pi_G \rightarrow \dot{\pi}_D \rightarrow \pi_G \rightarrow \dot{\pi}_C)^\omega$,
- $\mathcal{T}_{32}^2$: $\pi_C \rightarrow (\pi_G \rightarrow \dot{\pi}_D \rightarrow \pi_G \rightarrow \dot{\pi}_D)^\omega$.

The two teams have no shared information except positions, i.e. they do not know the other team's tasks and they do not communicate with each other. The position data is provided through the Vicon motion capture system and is used to emulate the sensing information that can be gathered using on board sensors.

5 RESULTS AND DISCUSSION

Figure 6 demonstrates the implementation of Example 1. The SPRK robots were divided into three groups, each assigned a synthesized LTS and a color to display (blue is $\mathcal{T}_{11}$, green is $\mathcal{T}_{12}$, and red is $\mathcal{T}_{13}$). The accompanying video shows that the swarm system satisfied the given specification and that there was no collision among all the robots as well as with the walls. In addition, we manually controlled a separate SPRK robot with a black marker to model a dynamic obstacle. When the dynamic obstacle has a velocity less than the maximum velocity of the swarm robots, we can guarantee that collisions do not happen.

Figure 7 shows the implementation of Example 2 in the Robotarium. Centralized and decentralized barrier certificates were used in

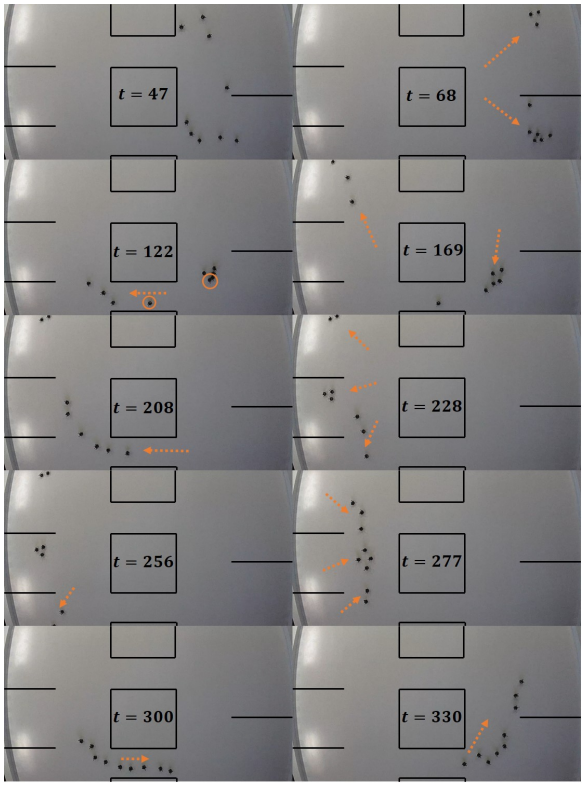

Figure 7: Example 2. Robots are divided into three groups. Lines represent walls in the workspace and dashed arrows indicate the motion.

two trials, where 9 robots starting at region C visited regions on two sides of the work space navigating through the corridor. For a large portion of the execution of the centralized controller, the swarm system satisfies the specifications and safety requirements. However, at time $t = 122s$, one robot stopped during the execution probably due to network delays or wheel slips, i.e., that robot could not receive the commands or it could not execute the command that it received. That failure caused collisions as the collision-free movement of the system relies on the correct execution of velocity commands by all the robots. After $t = 169s$, the failed robot started to move again and all robots moved correctly according to the specifications and barrier certificates. In contrast, the execution with the decentralized control barriers did not exhibit such collisions when a single robot failed to execute. This robustness is due to the fact that every single robot has its own barrier certificates and it will never run into an obstacle nearby as long as it is not experiencing significant delays in velocity execution. Thus, decentralized control performs better in case of individual failure as the input of each robot only depends on local sensing information, while in centralized control, control inputs of all robots are related.

Figure 8 shows the implementation of Example 3 on a swarm of SPRK robots. This example demonstrated two different teams of swarms sharing the same workspace, where team 1 consisted of 3 "fast" SPRKs with a velocity limit of $0.2m/s$ and team 2 had 4 "slow" SPRKs with a velocity limit of $0.1m/s$. Team 2 only patrolled between region C and D while team 1 visited four corner regions and went through region G. Note that there were no collisions and the specifications for both teams were satisfied. This demonstration implies that the synthesis scheme can be applied to swarm systems even if the swarms share the workspace with people or other types of robots but do not communicate with them.

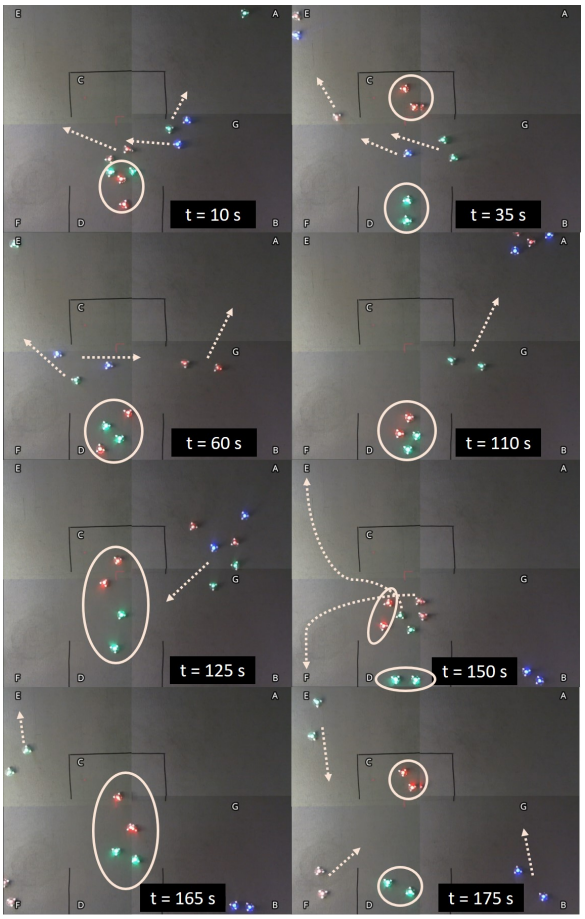

Figure 8: Example 3. Two separate SPRK swarms are carrying out their own tasks in a shared workspace. Lines represent walls in the workspace and dashed arrows indicate the motion. The slow team is circled out.

6 CONCLUSIONS

A controller synthesis framework from high-level specifications was implemented on different swarm systems. Control barrier certificates for collision avoidance were extended by considering polygonal obstacles in the environment. In addition, deadlock scenarios were avoided by using roadmaps. Demonstrations on different swarm robotic platforms show the flexibility and versatility of the proposed controller synthesis framework. In future work, we plan to extend the algorithms in this paper to enable swarms to perform tasks in dynamically changing environments.

7 ACKNOWLEDGMENTS

This research was supported by DARPA N66001-17-2-4058.

REFERENCES

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

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

[3] Aaron D Ames, Jessy W Grizzle, and Paulo Tabuada. 2014. Control barrier function based quadratic programs with application to adaptive cruise control. In *IEEE 53rd Annual Conference on Decision and Control (CDC)*. IEEE, 6271–6278.

[4] Jan Carlo Barca and Y. Ahmet Sekercioglu. 2013. Swarm robotics reviewed. *Robotica* 31, 3 (2013), 345–359. https://doi.org/10.1017/S026357471200032X

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

[6] Urs Borrmann, Li Wang, Aaron D Ames, and Magnus Egerstedt. 2015. Control barrier certificates for safe swarm behavior. *IFAC-PapersOnLine* 48, 27 (2015), 68–73.

[7] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. 2013. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence* 7, 1 (01 Mar 2013), 1–41. https://doi.org/10.1007/s11721-012-0075-2

[8] Edmund M Clarke, Orna Grumberg, and Doron Peled. 1999. *Model checking*. MIT press.

[9] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

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

[11] Markus Jager and Bernhard Nebel. 2001. Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 3. IEEE, 1213–1219.

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

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

[14] Marius Kloetzer and Calin Belta. 2010. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics* 26, 1 (2010), 48–61.

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

[16] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. 2009. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics* 25, 6 (2009), 1370–1381.

[17] Hadas Kress-gazit, Tichakorn Wongpiromsarn, and Ufuk Topcu. 2011. Correct, Reactive Robot Control from Abstraction and Temporal Logic Specifications. (2011).

[18] Salar Moarref and Hadas Kress-Gazit. 2017. Decentralized Control of Robotic Swarms from High-Level Temporal Logic Specifications. In *International Symposium on Multi-Robot and Multi-Agent Systems*. IEEE. To appear.

[19] Quan Nguyen and Koushil Sreenath. 2016. Exponential control barrier functions for enforcing high relative-degree safety-critical constraints. In *American Control Conference (ACC), 2016*. IEEE, 322–328.

[20] Petter Nilsson and Necmiye Ozay. 2016. Control Synthesis for Large Collections of Systems with Mode-Counting Constraints. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. ACM, 205–214.

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

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

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

[24] Jeff Shamma. 2008. *Cooperative control of distributed multi-agent systems*. John Wiley & Sons.

[25] Houshang H Sohrab. 2003. *Basic real analysis*. Vol. 231. Springer.

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

[27] Li Wang, Aaron D Ames, and Magnus Egerstedt. 2017. Safety Barrier Certificates for Collisions-Free Multirobot Systems. *IEEE Transactions on Robotics* (2017).

[28] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. 2012. Receding horizon temporal logic planning. *IEEE Trans. Automat. Control* 57, 11 (2012), 2817–2830.

[29] Tichakorn Wongpiromsarn, Alphan Ulusoy, Calin Belta, Emilio Frazzoli, and Daniela Rus. 2013. Incremental synthesis of control policies for heterogeneous multi-agent systems with linear temporal logic specifications. In *IEEE International Conference on Robotics and Automation*. IEEE, 5011–5018.

[30] Xiangru Xu, Paulo Tabuada, Jessy W Grizzle, and Aaron D Ames. 2015. Robustness of Control Barrier Functions for Safety Critical Control. *IFAC-PapersOnLine* 48, 27 (2015), 54–61.

[31] Yuan Zhou, Hesuan Hu, Yang Liu, and Zuohua Ding. 2017. Collision and Deadlock Avoidance in Multirobot Systems: A Distributed Approach. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2017).