Distributed Control of Robotic Swarms from Reactive High-level Specifications

Ji Chen, Ruojia Sun, and Hadas Kress-Gazit

***Abstract*— This paper presents a distributed strategy for automatically synthesizing controls for robotic swarms such that they achieve reactive, high-level formation tasks. In our framework, a user specifies formation and location-based swarm tasks, which may include reactions to environmental events, using linear temporal logic. Then, we synthesize a centralized finite automaton that represents the symbolic behavior of the swarm. To execute the automaton, we 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, we 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 we incorporate the constraints on sub-swarm sizes into the auction-based assignment algorithm. We demonstrate our control framework in simulation and with ten Anki-Vector robots in the lab.**

I. INTRODUCTION

Research in swarm robotics studies the coordination and interaction of large numbers of simple robots in terms of robustness, flexibility, and scalability [1]. Robot swarms have various potential applications such as surveillance [2], warehouse logistics [3], and collective construction [4]. While one direction of research on robot swarms focuses on creating simple single robot behaviors and analyzing the emergent behavior of the swarm [5]–[7] , recently researchers have been developing control schemes to achieve high-level swarm tasks in a top-down manner [8]–[11]. 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 [8], [9], or they lack reactivity to possible environment events [10], [11]. 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. 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 automatically synthesize controls for each

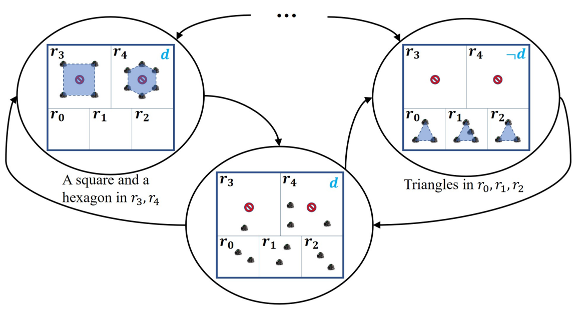


Fig. 1: Part of a finite automaton representing a formation-based reactive task where d is the environment event.

individual robot such that the swarm completes the task correctly. Our previous work [12] 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 [12]. 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.

Paper contributions. This paper 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.

II. RELATED WORK

Decentralized task assignment. There is a large body of work on decentralized task assignment for teams of robots (e.g. [13]–[16]). Some researchers focus on the communication mechanism that enables the robots to satisfy task

Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY, USA {jc3246,rs989, hadaskg} @cornell.edu. This work is supported by NSF IIS-1830471.

constraints: in [13], 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: [14] proposes a decentralized framework for multi-robot task assignment to satisfy LTL specifications under sensing uncertainties, and [15] shows a method using redundant robots for efficient task assignment with uncertain travel time. Unlike [14] that focuses on task allocation during single transitions in the policies, or [15] 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 [16]. The main difference of the assignment algorithm is that [16] 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 [17] 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 [18] 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 [19] and manipulators [20]). Recently, reactive synthesis has been studied in the context of robotic swarms [9] 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 [8] 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 [21]. Recently, [22], [23] 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 [24]. Then, in [12], 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 [22], [23] in that we address reactive tasks.

III. PRELIMINARIES

A. Abstractions for formation-based swarm tasks

In our previous work [12], 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 [12], 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. 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.

B. LTL Synthesis and Finite Automata

Given the propositions in Sec. III-A, we specify a high-level task as a linear temporal logic (LTL) formula φ, and use the algorithm in [18] 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 $L : Q \rightarrow 2^{X \cup Y}$ is a labeling function which maps a state to a set of propositions that are true in that state. Revisiting the example in section I, we synthesize an automaton $\mathbb{A}$ shown in Fig. 3a that satisfies the specification.

C. Market-based task assignment

Market-based approaches have been applied to decentralized task assignment for multi-robot systems [25]. The auction assigning method in this work is built on [16], 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.

IV. PROBLEM FORMULATION AND APPROACH

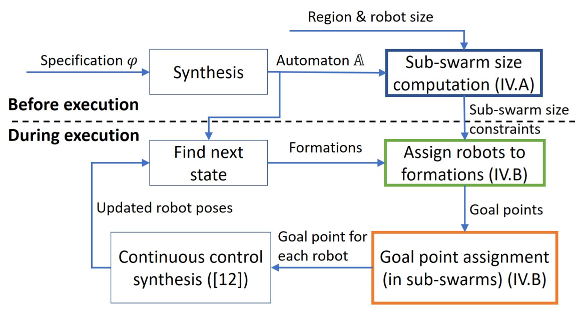

Region & robot size
Specification φ
Synthesis
Automaton $\mathbb{A}$
Sub-swarm size computation (IV.A)
Before execution
During execution
Sub-swarm size constraints
Find next state
Formations
Assign robots to formations (IV.B)
Updated robot poses
Goal points
Goal point for each robot
Continuous control synthesis ([12])
Goal point assignment (in sub-swarms) (IV.B)

Fig. 2: Flowchart of our control synthesis approach for high-level reactive swarm tasks. The contributions of this paper are in the boxes with the thick border.

This paper 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.

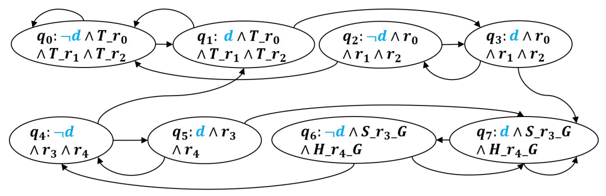
Our approach to the control synthesis problem is shown in Fig. 2. Before execution, users specify reactive, high-level tasks, from which we synthesize a finite automaton $\mathbb{A}$ [18]. We calculate constraints on the sub-swarm size for each transition in $\mathbb{A}$, based on the physical size of 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 [24]. 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.

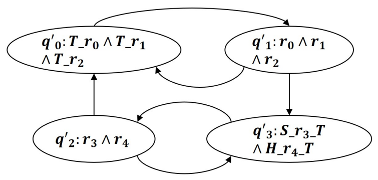
A. Computing sub-swarm size constraints

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 [12] 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 [12] 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 [23] which computes the minimum number of robots for non-reactive location-based tasks.


q_0: $\neg d \wedge T\_r_0 \wedge T\_r_1 \wedge T\_r_2$
q_1: $d \wedge T\_r_0 \wedge T\_r_1 \wedge T\_r_2$
q_2: $\neg d \wedge r_0 \wedge r_1 \wedge r_2$
q_3: $d \wedge r_0 \wedge r_1 \wedge r_2$
q_4: $\neg d \wedge r_3 \wedge r_4$
q_5: $d \wedge r_3 \wedge r_4$
q_6: $\neg d \wedge S\_r_3\_G \wedge H\_r_4\_G$
q_7: $d \wedge S\_r_3\_G \wedge H\_r_4\_G$

(a) Automaton $\mathbb{A}$ that satisfies the specifications in section I. Environment propositions are blue, robot propositions are black.


q'_0: $T\_r_0 \wedge T\_r_1 \wedge T\_r_2$
q'_1: $r_0 \wedge r_1 \wedge r_2$
q'_2: $r_3 \wedge r_4$
q'_3: $S\_r_3\_T \wedge H\_r_4\_T$

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

Fig. 3: (a) An automaton that implements the specifications in Section I. (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 I, shown in Fig. 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. 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. 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. 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 paper we determine $N^{q',min}_{r_i}$ and $N^{q',max}_{r_i}$ based on the procedure in [12].

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 $[N^{q',min}_{r_i}, N^{q',max}_{r_i}]$:

$$N^{q',min}_{r_i} \leq \sum_{\mathbf{r}_{p'}} x^{q',p'}_{r_i,\mathbf{r}_{p'}} \leq N^{q',max}_{r_i}, \tag{1}$$

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 $[N^{p',min}_{r_i}, N^{p',max}_{r_i}]$:

$$N^{p',min}_{r_i} \leq \sum_{\mathbf{r}_{q'}} x^{q',p'}_{\mathbf{r}_{q'},r_i} \leq N^{p',max}_{r_i}, \tag{2}$$

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').

$$\sum_{\mathbf{r}_{u'}} x^{u',q'}_{\mathbf{r}_{u'},r_i} = \sum_{\mathbf{r}_{v'}} x^{q',v'}_{r_i,\mathbf{r}_{v'}}, \tag{3}$$

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:

$$min \sum_{(q',p') \in \delta'} x^{q',p'}_{\mathbf{r}_{q'},\mathbf{r}_{p'}} \text{ subject to Eq.}(1),(2), and (3) \tag{4}$$

$$max \sum_{(q',p') \in \delta'} x^{q',p'}_{\mathbf{r}_{q'},\mathbf{r}_{p'}} \text{ subject to Eq.}(1),(2), and (3) \tag{5}$$

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 IV-B) 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. V-B).

B. 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. IV-A), assigns robots to different sub-swarms and goals. We execute the assignment algorithm twice for each transition in $\mathbb{A}$, as shown in Fig. 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 [16], 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 IV-A; 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 [16]) 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. 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 $\mathscr{N}$.

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 $\mathscr{N}$ (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:** $T^*$, $N_t^{min}$, $N_t^{max}$ $\forall t \in T^*$, $\mathscr{N}$
**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:      $(n_{T^*}^{min}, n_{T^*}^{max}) := updateAvailabilities(\mathscr{N})$
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:     $(\mathscr{N}_t, B) := findNeighborBids(t, \mathscr{N})$
11:     **if** $\mathscr{N}_t = \emptyset$ or $(\mathscr{N}_t \neq \emptyset$ and $b > max_{i \in \mathscr{N}_t}(b_i \in B))$ **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** $b_i = max_{i \in \mathscr{N}_t}(b_i \in B)$ **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 [16]).

V. DEMONSTRATION AND EVALUATION

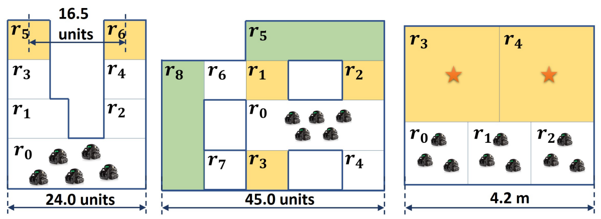


Fig. 4: Workspace of examples in Sec V. From left to right: small environment, complex task, physical demonstration.

A. Simulation - small environment

1) Task description: In the workspace shown in Fig. 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.

2) High-level synthesis: We synthesize an automaton $\mathbb{A}$, using the approach in section III-B, 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 IV-A, we calculate the bounds on the number of robots for these two sub-swarms as $[4,36]$ and $[6,36]$ respectively.

3) 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. 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 [12]. 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.

Travel distance: The red lines in Fig. 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 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

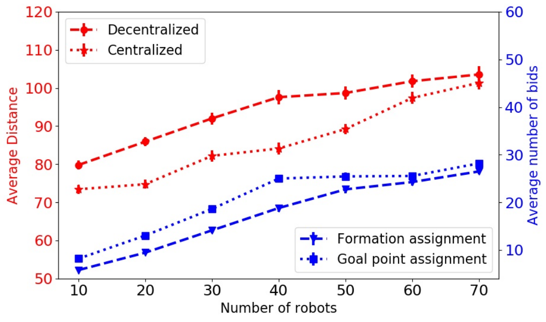


Fig. 5: Comparison between centralized and decentralized assignment on travel distances (red) and average number of bids needed for robots per transition for the decentralized approach (blue) with respect to the swarm size.

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. 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.

B. Simulation - complex task

1) Task description: In the workspace shown in Fig. 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.)

2) 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

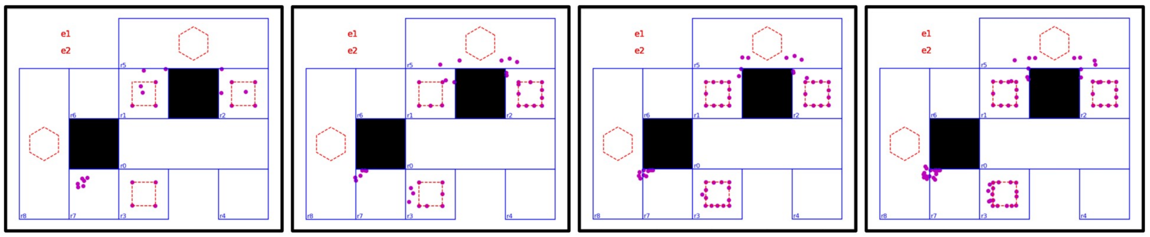


Fig. 6: Different sizes of simulated swarms performing the same transition (both e_1 and e_2 are true) in $\mathbb{A}$ (Sec. V-B). Swarm sizes are 24, 40, 56, and 72 from left to right. Robots are purple circles, intended formations are displayed in red.

(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. 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.

C. Physical demonstration

1) Task description: We demonstrate our approach using 10 Anki-Vector robots [26] 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 (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.

2) 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.

3) Demonstration: Snapshots of the demonstration are shown in Fig. 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. 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.

VI. CONCLUSION AND 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 [27]).

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 [28] to speed up the execution and avoid deadlocks since each robot plays a cooperative role in the swarm to achieve the task.

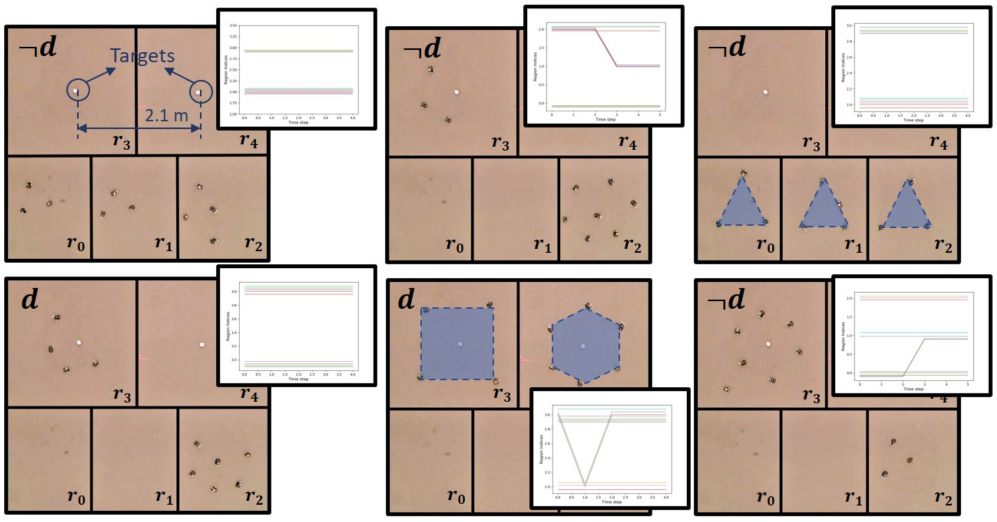


Fig. 7: Snapshots of 10 Anki Vectors accomplishing a high-level task. The plots represent the region selections of robots over time during the assignment. Each line represents the region selection of one robot over the number of auctions.

References

[1] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, no. 1, pp. 1–41, 2013.

[2] M. Saska, V. Vonásek, J. Chudoba, J. Thomas, G. Loianno, and V. Kumar, "Swarm distribution and deployment for cooperative surveillance by micro-aerial vehicles," *Journal of Intelligent & Robotic Systems*, vol. 84, no. 1, pp. 469–492, 2016.

[3] A. Farinelli, E. Zanotto, E. Pagello, *et al.*, "Advanced approaches for multi-robot coordination in logistic scenarios," *Robotics and Autonomous Systems*, vol. 90, pp. 34–44, 2017.

[4] K. H. Petersen, N. Napp, R. Stuart-Smith, D. Rus, and M. Kovac, "A review of collective robotic construction," *Science Robotics*, vol. 4, no. 28, 2019.

[5] O. Soysal and E. Sahin, "Probabilistic aggregation strategies in swarm robotic systems," in *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.* IEEE, 2005, pp. 325–332.

[6] T. H. Labella, M. Dorigo, and J.-L. Deneubourg, "Division of labor in a group of robots inspired by ants' foraging behavior," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 1, pp. 4–25, 2006.

[7] J. Bachrach, J. Beal, and J. McLurkin, "Composable continuous-space programs for robotic swarms," *Neural Computing and Applications*, vol. 19, no. 6, pp. 825–847, 2010.

[8] M. Kloetzer and C. Belta, "Temporal logic planning and control of robotic swarms by hierarchical abstractions," *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 320–330, 2007.

[9] S. Moarref and H. Kress-Gazit, "Reactive synthesis for robotic swarms," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2018, pp. 71–87.

[10] I. Haghighi, S. Sadraddini, and C. Belta, "Robotic swarm control from spatio-temporal specifications," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 5708–5713.

[11] F. Djeumou, Z. Xu, and U. Topcu, "Probabilistic swarm guidance subject to graph temporal logic specifications," in *Robotics: Science and Systems (RSS)*, 2020.

[12] 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)*, 2020, pp. 8027–8034.

[13] H.-L. Choi, A. K. Whitten, and J. P. How, "Decentralized task allocation for heterogeneous teams with cooperation constraints," in *Proceedings of the 2010 American Control Conference*. IEEE, 2010, pp. 3057–3062.

[14] P. Schillinger, M. Bürger, and D. 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)*. IEEE, 2018, pp. 7330–7337.

[15] A. Prorok, "Robust assignment using redundant robots on transport networks with uncertain travel time," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 4, pp. 2025–2037, 2020.

[16] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas, "Distributed multi-robot task assignment and formation control," in *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2008, pp. 128–133.

[17] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1989, pp. 179–190.

[18] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive (1) designs," *Journal of Computer and System Sciences*, vol. 78, no. 3, pp. 911–938, 2012.

[19] S. Maniatopoulos, P. Schillinger, V. Pong, D. C. Conner, and H. Kress-Gazit, "Reactive high-level behavior synthesis for an atlas humanoid robot," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 4192–4199.

[20] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Reactive synthesis for finite tasks under resource constraints," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 5326–5332.

[21] M. Kloetzer, X. C. Ding, and C. Belta, "Multi-robot deployment from ltl specifications with reduced communication," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*. IEEE, 2011, pp. 4867–4872.

[22] S. Moarref and H. 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)*. IEEE, 2017, pp. 17–23.

[23] ——, "Automated synthesis of decentralized controllers for robot swarms from high-level temporal logic specifications," *Autonomous Robots*, vol. 44, no. 3, pp. 585–600, 2020.

[24] J. Chen, S. Moarref, and H. Kress-Gazit, "Verifiable control of robotic swarm from high-level specifications," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018, pp. 568–576.

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

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

[27] F. Amigoni, J. Banfi, and N. Basilico, "Multirobot exploration of communication-restricted environments: A survey," *IEEE Intelligent Systems*, vol. 32, no. 6, pp. 48–57, 2017.

[28] H. Wang and M. Rubenstein, "Shape formation in homogeneous swarms using local task swapping," *IEEE Transactions on Robotics*, vol. 36, no. 3, pp. 597–612, 2020.