Shape Formation in Homogeneous Swarms Using Local Task Swapping

Hanlin Wang and Michael Rubenstein

***Abstract*—The task of shape formation in robot swarms can often be reduced to two tasks—assigning goal locations to each robot and creating a collision-free path to that goal. In this article, we present a distributed algorithm that solves these tasks concurrently, enabling a swarm of robots to move and form a shape quickly and without collision. A user can specify a desired shape as an image, send that to a swarm of identically programmed robots, and the swarm will move all robots to goal locations within the desired shape. This algorithm was executed on a swarm of up to 1024 simulated robots and a swarm of 100 real robots, showing that it reliably converges to all robots forming the shape.**

***Index Terms*—Distributed robot systems, multirobot systems, swarms.**

I. Introduction

SHAPE formation is one of the fundamental problems in swarm systems. The task is often framed as moving a set of robots, which are initially located randomly in space, into a given arbitrary target formation, which is often specified by a set of locations. It plays an important role in a wide variety of applications, such as modular robots [1], warehouse management [2], entertainment applications [3], and more [4].

In general, the complete shape formation problem can be divided into two subproblems—assignment of robots' locations in shape and formation control.

The assignment subproblem tries to divide the goal locations among the individuals, often in an optimal way, such as minimizing the total distance traveled by the swarm. This problem has been well studied and there are several algorithms which can find the optimal assignment, including the Hungarian algorithm [5], auction algorithms [6], [7], and iterative methods [8], [9]. Recent work shows that some certain assignments which minimize a cost of interest can help to reduce the computational complexity of path planning problems. One typical cost of interest used is the sum of the distance traveled by all agents [10]–[12], and the other cost of interests are the sum of the square distance traveled [13] and the maximal distance traveled [14], which help to minimize the total time elapsed. In these methods, the calculation for the assignment is handled by a centralized coordinator. These centralized strategies can deliver a solution to the assignment problem, but do not easily scale to large numbers of robots, present a single point of failure, and do not easily adapt to situations where the number of robots is unknown or can vary.

Unsurprisingly, the distributed assignment methods, on the other hand, can often scale well to the number of robots, and can be more robust to failures [15] and varying numbers of robots. Past efforts try to solve the distributed assignment problem by following an incremental distributed refinement process [16], [17]. Here, the order how agents explore each goal has significant effect on convergence rate. In [16], the agents follow a preassigned order which assures that a correct assignment of agents

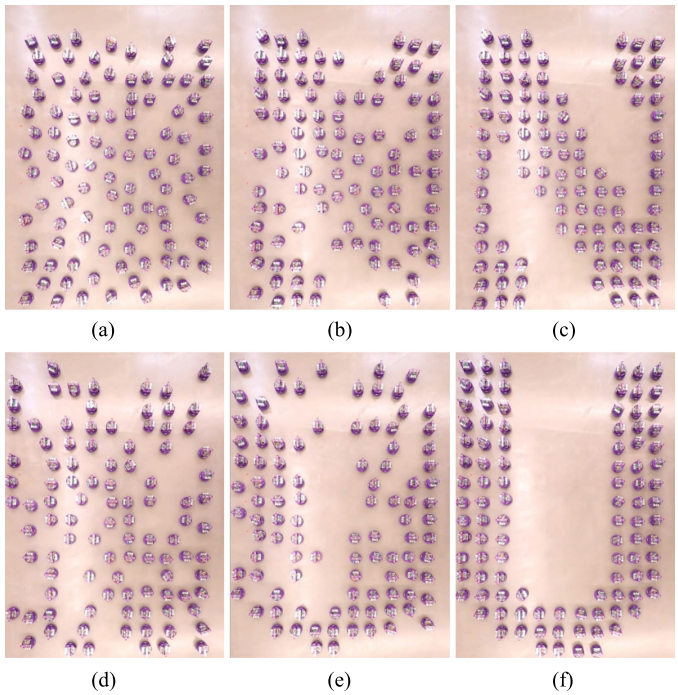


Fig. 1. Still images from a 100 robot shape formation experiment. The robots start in a random configuration, and move to form the desired "N" shape. Once this shape is formed, they then form the shape "U." The entire sequence is fully autonomous using the distributed algorithm described in this article. (a) T = 0 s. (b) T = 20 s. (c) T = 64 s. (d) T = 72 s. (e) T = 80 s. (f) T = 112 s.

Manuscript received November 6, 2019; accepted January 7, 2020. This article was recommended for publication by Associate Editor N. Gans and Editor P. Robuffo Giordano upon evaluation of the reviewers' comments. This work was supported by the Alfred P. Sloan Research Fellowship. *(Corresponding author: Michael Rubenstein.)*

H. Wang is with the Department of Computer Science, Northwestern University, Evanston, IL 60601 USA (e-mail: h.w@u.northwestern.edu).

M. Rubenstein is with the Department of Computer Science and the Department of Mechanical Engineering, Northwestern University, Evanston, IL 60601 USA (e-mail: rubenstein@northwestern.edu).

This article has supplementary downloadable material available at https://ieeexplore.ieee.org provided by the authors. This includes a video file, which demonstrates the described algorithm for shape formation running on a real 100 robot swarm and a 1024 robot simulation. This material is 6.10 MB in size.

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2020.2967656

to tasks is always achieved after exploring at most a polynomial number of assignments. In [17], authors obtained an efficient convergence by forcing agents to follow a certain path. This path is collision-free when agents have infinitely small size, but when agents have finite size, the path cannot provide a collision-free guarantee.

After determining the role in shape, each agent then needs to move cooperatively to form the desired shape. In the past, many methods to produce the formation have been proposed. According to the types of actively controlled variables [18], such as agent's position or distances to the neighbors, formation control methods can be categorized into local measurement-based methods [19]–[31] and position-based methods [10]–[14], [32]–[50].

In local measurement-based methods [19], the agents form the desired shape by actively controlling its distance [20]–[24], bearing [25], or both [26]–[30], relative to its neighbors. This type of methods only require the use of relative measurements; therefore can be employed in the GPS-denied environments, e.g., indoor environments. For local measurement-based methods, the challenge is how to obtain the global stabilization to the desired formation using only peer-to-peer information [26].

Some methods achieve the global stabilization by relying on the leader agents [24], [28]–[30]. In these methods, the leader tracks its desired trajectory, and the nonleader agents are tasked to maintain certain graph structures rooted from the leader agent where each vertex characterizes an agent and each edge characterizes an interagent measurements, such as distance or relative position. These methods allow the swarm to stabilize to a formation that is even dynamically moving, but require an additional leader selection phase to assign a role (leader or nonleader) to each agent.

The methods proposed in [26] and [27] are leaderless, these methods enable a group of agents to reliably produce a rigid shape, using the relative positions of agent's neighbors. Nevertheless, in order to achieve global stability, the method proposed in [27] needs the communication graph among agents to be complete, and the method proposed in [26] requires the desired formation to satisfy some specific topological conditions.

While local measurement-based methods permit operations in GPS-denied environments, they often require a centralized coordinator, or the use of a complete communication network, to initially assign each robot a position in final shape. Moreover, without any additional mechanism, it often fails to provide an absolute collision avoidance guarantee when agents have finite size.

To the contrary, in position-based methods, the desired formation is achieved by actively controlling the agents' positions. This type of methods require that each agent is able to measure their own positions with respect to a global coordinate system. Here, the challenge is how to efficiently generate collision-free trajectories where agents can achieve the desired formation by moving to goal locations.

Some previous work tackled the problem in a discrete setting [32]–[35], while others solved the problem in a continuous setting [36]–[39]. As expected, these centralized methods suffer from the curse of dimensionality (because the dimension of swarm's joint configuration space increase exponentially over swarm size), hence often cannot easily scale to large-scale swarms, such as a swarm of over 1000 agents.

An alternative method is to use an artificial potential function to guide agents to the desired formations using gradient descent. Some authors make use of gradient descent to drive agents to goals [46], [47], and some use the potential function to modify current trajectories locally to prevent collision and maintain connectivity [48]. The drawbacks for this kind of method are that it may take a long time to converge and there is no guarantee provided that they can form the desired shape [11], [49].

Distributed multiagent path planning is a well studied topic. Some methods are based on local measurements, either relative velocities [40]–[42] or relative positions [43], but none of these methods can provide a deadlock-free guarantee, agents can get stuck in a situation where no action can be made for further progress, yet the shape is incomplete. In fact, in our review of distributed path planners [31], [40]–[43], [45], none of the methods can provide a deadlock-free guarantee and absolute collision-free guarantee at the same time. This is also suggested in [43], which also claims that no deadlock-free distributed path planner that is with absolute collision-free guarantee exists. Other approaches make use of the communication among agents, but in order to guarantee the correctness of the method, require either a lossless fully connected network [44] or precise velocity control [11], which can be difficult to guarantee when implemented in a physical system. In [45], authors presented a distributed collision avoidance strategy which can resolve some certain types of deadlocks using local communication only, but the method cannot resolve all types of deadlocks. A distributed receding horizon control based method is proposed in [31], this method requires only the use of relative sensing in robot's local coordinate frames, and is able to provide mathematical guarantees on the achievement of the rendezvous, however, it has not been shown that the method can provide collision-free guarantee and absolute deadlock-free guarantee at the same time.

In this article, we present a fully distributed shape formation algorithm where each agent is identically programmed and takes the same input, a set of goal points that describes the desired shape. Each agent will use local communication to actively refine the goal assignment and control its position in a distributed fashion. To the best of our knowledge, and supported by [43], our algorithm is the first provably correct fully distributed shape formation algorithm that can also provide absolute collision-free and deadlock-free guarantees, requiring only the use of local communication. Moreover, the physical experiments and simulations presented show that our algorithm is robust to real-world nonidealities, such as communication errors, sensing errors, and imperfect robot motion.

II. Problem Definition

In this article, we propose an algorithm that when given a set of desired target points (which are described by a set of nodes on a grid), moves a swarm of mobile agents so that each agent is located at a target position, and no target position has more than one agent. For each agent, the set of target positions

are known *a priori*, moreover, all the agents agree on the same global reference frame. This algorithm must distribute the target positions among the agents and then drive the agents to their corresponding target position without collision. The system is distributed, agents are identically programmed, and act based on local information gathered through communication. In this section, we will formally state the problem and introduce the notations used in this article.

A. Agent Model

Agents are modeled as two-dimensional omnidirectional robots equipped with position and orientation sensing, i.e., each agent can measure its own position and orientation in a global coordinate system at all times. Each agent is treated as a circle with a finite radius r and can move in any direction at speed v_m, or stop. Additionally, each agent is able to communicate with any agent lying within its communication range $R \geq 4\sqrt{2}r$. To simplify the analysis and description, we assume the following.

a) Each agent has the same clock frequency f_{clock}.
b) Each agent is able to constantly transmit messages to the neighbors in communication range at a fixed rate f_{comm}.
c) The local interagent communication is lossless.
d) Each agent has the same v_m.
e) The communication latency is negligible.

Note that here we do not have any assumption on the phase of the agent's clock relative to each other, they can be asynchronous in phase. When the algorithm is implemented in the real world, these assumptions can be relaxed to accommodate the real-world nonidealities, see Section V-B3 for detailed discussion.

B. Notations

For the sake of describing our algorithm and formulating the problem, we introduce the notations as follows.

Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of agents, where each agent $a_i \in A$ has a position $p_{a_i}(t) \in \mathbb{R}^2$ at time t. For all $p \in \mathbb{R}^2$, p^x, p^y denote p's x and y components, respectively. $||\cdot||$ denotes the Euclidean norm on $\mathbb{R}^2$ space and $\succ$ denotes the lexicographic order on $\mathbb{R}^2$ space, namely, $p_1 \succ p_2$ if and only if: $p_1^x > p_2^x$, or $p_1^x = p_2^x$ and $p_1^y > p_2^y$. Let $Q = \{q_1, \ldots, q_m\}$ be a set of distinct target locations, where $q_i \in \mathbb{R}^2$, we assume that $\forall q_i, q_j \in Q, ||q_i - q_j|| \geq 2\sqrt{2}r$, i.e, in the desired shape no pair of robots collide with each other. Moreover, we use $T_{a_i}(t) \in Q$ to denote a_i's assigned target position at time t. We assume that every agent has the same communication range R, and $N_{a_i}(t) \subset A$ denotes the set such that at time t, $\forall a_j \neq a_i$, $||p_{a_i}(t) - p_{a_j}(t)|| \leq R$ if and only if $a_j \in N_{a_i}(t)$, in the other words, $N_{a_i}(t)$ is the set of agents that are able to communicate with a_i at time t.

C. Problem Formulation

Our task is to design an algorithm to move a swarm of n identical robots, represented by set A, from their initial positions to an arbitrary connected target formation, represented by set Q. To simplify the problem, we assume $|Q| = |A|$. The algorithm should be deadlock-free and collision-free, i.e.,

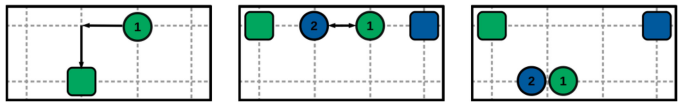


Fig. 2. Illustration of the grid discretization of space and possible collision cases. The intersections of grey dashed lines represent the feasible waypoints, and agents travel on the edges between waypoints. Each agent's position is shown with a colored circle and its goal point is shown with a square of the same color. Moreover, we label each agent with a unique number and use the arrow to show agent's incentive for next step. (left) A valid trajectory for a single agent to move to its goal. The trajectory is shown as a sequence of arrows. (middle) An edge collision, where blue and green robots both intend to travel on the edge in black, in opposite directions. Here neither can make progress without collision. (right) Collision happens on a waypoint, where the blue and green robots try to move to the same waypoint at the same time, physically colliding.

- $\exists t_{\max} > 0$ such that at any time $t > t_{\max}$, it holds that: $\forall a_i \in A$, $p_{ai}(t) = T_{a_i}(t)$, moreover $\forall a_j \neq a_i, T_{a_j}(t) \neq T_{a_i}(t)$,
- $\forall t \geq 0$, for any two agents $a_i \neq a_j$, $||p_{a_i}(t) - p_{a_j}(t)|| \geq 2r$.

III. Approach

To form the goal shape, each agent needs to pick a valid goal, and then move on a collision-free and deadlock-free path toward that goal without any centralized coordination. For this task, two subproblems arise. One of them is solving duplicated assignments, which is caused by the limited sensing ability of the agents. Agents have limited communication range so they have to determine their targets based only on the local information. This makes it possible that there exist multiple robots holding the same target. The other subproblem is planning each robot's motion based on local information so as to generate collision-free and deadlock-free paths toward the goals. Additionally, if every agent's target is unique, the motion planner should guarantee that each agent will reach the target in a finite amount of time.

In our method, the task is handled by two modules—the *new goal selector*, which is used to pick a valid goal, and the *motion planner*, which is used to plan the agent's motion. Each agent uses the *motion planner* to move to its current goal, and if it encounters another agent holding the same goal, it uses the *new goal selector* to pick another goal. A detailed description is shown as Algorithm 1. Note that this algorithm runs on each agent of the swarm.

A. Motion Planning

To generate a collision-free path, we first convert the continuous environment into a discrete grid, as shown in Fig. 2. While this grid representation of the environment will make agent's motion less efficient, it helps to reduce the computational cost of motion planning. Note that the grid here is the same grid goal points are located on. Let l be the length of the grid edge, where the constraint that $2\sqrt{2}r \leq l \leq \frac{R}{2}$ is enforced for the purpose of collision avoidance. Furthermore, we assume that there are no obstacles located in the environment. With this representation, each agent's path is given by a sequence of the waypoints, i.e., the nodes of the grids.

For every two adjacent waypoints, the motion controller enforces that the robots travel on the line segment between them.

Algorithm 1: General Framework for Shape Formation.

Input: $Q = \{q_1, q_2, ..., q_m\}$
$T_{a_i}(t) \leftarrow$ random element in Q
while *True* **do**
  if $\exists a_j \in N_{a_i}(t)$, *s.t.* $T_{a_j}(t) = T_{a_i}(t)$ **then**
    run *new goal selector* (Alg. 4 Line 3-10)
  run *motion planner* (Alg. 2 Line 25-29)

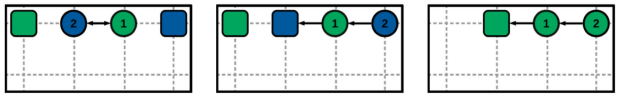


Fig. 3. Illustration of possible cases where an agent may change its goal. All the information is encoded in the same way as Fig. 2. (left) For any pair of agents located within each other's communication range, if goal swap can help them to reduce the pairwise total distance traveled (in term of Manhattan distance), then goal swap occurs. (middle) If the goal swap does not affect the total pairwise travel distance, these two agents randomly decide whether to swap. (right) If both agents hold the same goal, one of them will run the *new goal selector* algorithm to select a new goal from Q.

The motion controller plans every robot's motion, such that the following constrains are satisfied.

Constraint III-A.1: At any time *t*, no agent moves to the waypoint that is currently occupied by the other, and no pair of agents move toward the same waypoint at the same time.

Constraint III-A.2: At any time *t*, no pair of agents travel on the same edge in opposite directions.

For each agent located at any waypoint, there are five possible actions—move $north$, $east$, $south$, $west$, and $wait$. Agents should choose the action that greedily reduces the Manhattan distance to its goal point. Once the agent determines its next action, and if this action is not $wait$, it first uses communication to check whether the waypoint is occupied by any other agent. If it is occupied, the agent executes $wait$ and continues using communication to check the availability of the waypoint (Algorithm 2, Lines 26–27). If there is no other agent occupying this waypoint, the agent then starts to check if any other agent wants to move to the same waypoint as it does. If there are multiple robots intending to go to the same waypoint (x, y) at the same time t, then the robot a_i whose current position $p_{ai}(t)$ is the lexically largest will go first (Algorithm 2, Lines 28–29).

As the agent moves toward its goal, it continually tries to improve its goal assignment, changing its goal based on local information. When it senses a neighbor with whom a swapped goal would result in a reduced pairwise traveled distance (in terms of Manhattan distance), it swaps goals with that neighbor (Algorithm 4, Lines 11–15). If swapping goals with a neighbor does not change the pairwise distance traveled, they swap goals with a probability $0 < \beta < 1$ (Algorithm 4, Lines 16–21). When a goal conflict is sensed, i.e., a neighbor is seen that is holding the same goal point, one of the agents picks a new goal from Q to eliminate the duplicated goal (Algorithm 4, Lines 3–10). An illustration of the cases in which an agent may change its goal are shown in Fig. 3.

It is possible that multiple agents (more than two agents) intend to swap the goals at the same time, for example, at time t, a_i intends to swap the goal with a_j while a_j intends to swap the goal with the third agent a_k. In our implementation, the pairwise goal swap is achieved by a 2-way handshake (Algorithm 4, Lines 12 and 18). Only the pair of who successfully handshake with each other can swap the goal. To be specific, when agent a_i intends to swap the goal with a_j, if $p_{a_j}(t) \succ p_{a_i}(t)$, then it will take the role of *client* in this handshake, otherwise if $p_{a_i}(t) \succ p_{a_j}(t)$, a_i will act as *server* in the handshake. A *client* agent a_i will send a handshake request to its intended goal swap peer a_j, which is a handshake *server* since $p_{a_j}(t) \succ p_{a_i}(t)$, and then wait for the acknowledgement (ACK) from this *server* agent a_j for certain amount of time. On the other hand, a *server* agent a_j will wait for the handshake request from its intended goal swap peer a_i for certain amount of time, which is a *client* agent since $p_{a_j}(t) \succ p_{a_i}(t)$, and send back an ACK to a_i after receiving the handshake request from a_i. Note that it is possible that a *server* agent a_j receives the handshake requests from other agents that is not its intended goal swap peer. When this happens, it will answer the requests from these agents with a negative ACK (or does not answer these requests at all so as to trigger the handshake timeout). The *client* agent a_i will update its goal only after receiving the ACK from the intended goal peer a_j, and the *server* agent a_j will update its goal after receiving the handshake request from its intended goal swap peer a_i.

B. New Goal Selecting

As we want each agent to have a unique goal in the end, the algorithm needs to eliminate any duplication in the assigned goals. For the *new goal selector* algorithm, we desire the swarm to have as many different goals assigned as possible. When the total number of assigned goals is equal to the size of the swarm, no pair of agents will have the same goal. This implies that the *new goal selector* should keep the number of assigned goals growing. Therefore, every time a new goal is selected, the total number of assigned goal points should be nondecreasing.

1) Random Selector: A simple *new goal selector* would be a random selector. For every pair of agents a_i, a_j that detect that they hold the same goal, if $p_{a_j}(t) \succ p_{a_i}(t)$, where $\succ$ denotes the lexical order on $\mathbb{R}^2$ space, then a_i randomly picks a new goal from the set Q.

While the random selector can guarantee the process to be almost surely convergent, the probability of picking an unassigned goal will decay over the number of assigned goals, leading to a relatively long convergence time. We therefore introduce a heuristic to speed convergence.

2) Gradient-Based Selector: The gradient algorithm, which is a well-known collective behavior, also known as hop-count algorithm [51], [52], can be adapted to improve goal selection. It is a simple algorithm that involves two agent roles, the common agent and the anchoring agent, and both roles transmit a single message containing a position q_u and a hop-count h. Each common agent listens to messages from its neighbors in communication range R, finds the message with the lowest hop-count received, (q_u^x, q_u^y, hop), and then transmits the message $(q_u^x, q_u^y, hop + 1)$ (Algorithm 2, Lines 17–19).

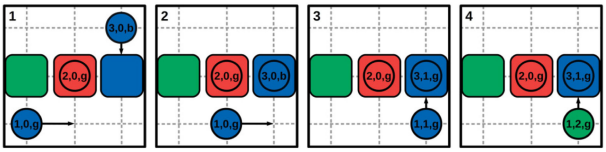


Fig. 4. Example of agents using gradient-based selector to update their goals. Goal positions are shown as a colored square, agent positions are shown as a circle whose color matches its current goal. Agents are labeled with its index i, hop-count value hop, and candidate goal q_u color (r-red, b-blue, g-green), respectively. For this example, we assume that each agent's communication range is one grid length. Initially, in *frame 1*, the goal T, for agents a_1 and a_3 is blue, and a_2 is red. In *frame 2,* agents a_1 and a_3 move toward their goal, with a_3 arriving at its goal. In *frame 3*, the hop-count is updated for agents and a_1 continues toward its current goal. In *frame 4*, agent a_1 sees a_3 with the same goal, and since $a_3 \succ a_1$, a_1 changes goals, choosing the goal indicated by the hop-count message.

The basic hop-count algorithm from [51] and [52] can be modified in the following way to allow for better goal selection. An agent will take on the anchoring role when it is one grid length away from an unassigned goal point (q_u^x, q_u^y) (by "unassigned goal" we mean the goal that is not assigned to any of the agent's neighbors in communication range), and transmit the message $(q_u^x, q_u^y, 0)$ (Algorithm 2, Lines 20–24). An anchoring agent will become a common agent when it no longer detects a unassigned goal that is one grid length away. Every agent a_i keeps the latest goal point (q_u^x, q_u^y) it transmitted as the candidate goal q_u. When a_i detects that there is another agent a_j holding the same goal and $p_{a_j}(t) \succ p_{a_i}(t)$, it then uses the current candidate goal q_u to update its goal $T_{a_i}(t)$ (Algorithm 4, Lines 3–10).

This gradient-based selector helps to prevent an agent from selecting a goal that is already occupied, while also propagating information about valid goals throughout the entire swarm. This helps increase the possibility (compared to random goal selector) that the new goal generated from "new goal selector" is valid, i.e., the goal has not been occupied by other agents yet. See Fig. 4 for a graphical illustration.

In addition, the swarm can also use the gradient hop-count to detect whether the shape is completed. If the shape is completely formed, there will be no anchor nodes in the swarm, so the gradient value of each agent will increase temporally. If any agent holds a gradient value larger than the number of agents, then the agent knows there are no anchor nodes, and therefore the shape is completed. Once any agent detects that the shape is complete, it will send a message which propagates across the entire swarm telling every other agent that the shape is complete.

Algorithm 2: Main Component.

Input: $Q = \{q_1, q_2, ..., q_m\}$, β.

```
/* Q is the set of goal points, β is a constant where 0 < β < 1.  */
1  wp ← current waypoint
2  hop ← ∞
3  q_u ← random element in Q
4  T ← random element in Q
5  next_step ← current waypoint
6  Δt ← 2 / f_comm
7  last_check ← clock()
8  while True do
9      surroundings ← {(wp^x − l, wp^y), (wp^x, wp^y − l), (wp^x +
         l, wp^y), (wp^x, wp^y + l)} /* the set of waypoints one grid length
         away from current waypoint. */
10     wait_flag ← 0
11     for i in surroundings do                    /* find the next waypoint */
12         if choice of i reduces distance to goal then
13             next_step ← i
14             Break
15     msg_buff ← all messages received since clock() - Δt
16     if msg_buff is not empty then
           /* find the message that contains the lowest hop-count value. */
17         msg_min ← the message in msg_buff that contains the lowest
             message.hop
18         hop ← 1 + msg_min.hop
19         q_u ← msg_min.q_u
20         for i in surroundings do /* check if there is any unassigned goal
             one grid length away */
21             if i ∈ Q and ∀ msg_j in msg_buff, msg_j.T ≠ i then
22                 q_u ← i
23                 hop ← 0
24                 Break
25         for i in msg_buff do /* loop through all the messages in msg_buff
             to check if there is any potential collision */
26             if i.wp == next_step then
27                 wait_flag ← 1          /* next_step is occupied */
28             if i.next_step == next_step and i.p ≻ p then
29                 wait_flag ← 1          /* other agent with higher priority
                   intends to go to the same waypoint */
30     if wait_flag == 0 and clock() - last_check > Δt then   /* there is no
         potential collision and the agent has stay at the current waypoint
         long enough */
31         wp ← next_step
32         agent moves to wp
33         last_check ← clock()
34     else
35         stay at current waypoint
```

Algorithm 3: Broadcast Component.

```
1  while True do
       /* Forge the message to be transmitted, the message contains: agent's
          current position, current waypoint, next waypoint, current goal,
          candidate goal, and hop-count value.                          */
2      msg ← {p, wp, next_step, T, q_u, hop}
3      transmit msg
4      sleep 1 / f_comm
```

C. Implementation

In this section, we describe the implementation of shape formation algorithm using gradient-based selector in detail. The algorithm consists of three components—main component, broadcast component, and goal manager. The main component coordinates agent's motion based on the information coming from its neighbors in communication range so as to avoid collision; the broadcast component constantly transmits messages to neighbors at a fixed frequency f_{comm}, and the neighbors in communication range will use this information to coordinate their traffic; the goal manager refines agent's assigned goal so as to eliminate the duplicated assignment and resolve the deadlocks. These three components can be implemented using three separate threads running on each agent that communicate through shared memory. The sketches of these three modules are shown in Algorithms 2–4. Note that all the variables are thread-public.

1) Main Component: In main component, the agent has two tasks—use the messages from its neighbor to plan its motion (Algorithm 2, Lines 25–29), and perform the gradient algorithm so as to help to propagate the information about unassigned goal

Algorithm 4: Goal Manager.

```
1  while True do
       /* We use a_i to denote the agent that is executing this thread */
2      if receive a msg_in from any other agent a_j then
3          if a_j holds the same goal then
4              if p_{a_j} ≻ p_{a_i} then
5                  if rand(0, 1.0) > 0.1 then
6                      T ← q_u
7                      last_check ← clock() /* The goal changes, as a
                         result, Alg. 2 Line 11-13 may change next_step,
                         hence we need to reset the timer for safety
                         checking so as to avoid collision */
8                  else
9                      T ← random element in Q
10                     last_check ← clock() /* Same reason as Alg. 4
                         Line 7. */
11         if the goal swap with a_j can reduce cost then
12             Execute the 2-way handshake with a_j
13             if 2-way handshake succeeds then
14                 updates agent's goal
15                 last_check ← clock() /* Same reason as Alg. 4 Line 7.
                     */
16         if the goal swap with a_j doesn't effect cost then
17             if rand(0, 1.0) < β then
18                 Execute the 2-way handshake with a_j
19                 if 2-way handshake succeeds then
20                     updates agent's goal
21                     last_check ← clock() /* Same reason as Alg. 4
                         Line 7. */
```

through the swarm (Algorithm 2, Lines 15–24). The variables and system calls that are used in this component are as follows:

- *hop:* Agent's current hop-count value;
- q_u*:* Agent's candidate goal;
- T*:* Agent's current goal, i.e., the goal that the agent is moving toward;
- *wp:* The waypoint that agent is claiming, i.e., the waypoint that agent is currently moving to or staying at;
- p*:* The agent's position;
- *next_step:* Agent's next waypoint;
- Δt*:* The amount of time such that: If the agent tries to plan its motion at time t, then it will use all the messages received between $t - \Delta t$ and t to do the calculation;
- *clock():* The system call that returns the time elapsed since the program started;
- *last_check:* The variable to record the time when the agent arrived at current waypoint;
- *surroundings:* The set of four waypoints that are one grid length away from current waypoint;
- *msg_buff:* The set of messages that are received in the last Δt amount of time;
- *wait_flag:* The flag variable that helps agent to check the potential collisions, specifically, if *wait_flag* is 0, then agent can move to the next waypoint; otherwise if *wait_flag* is 1, then the agent needs to stay at the current waypoint.

Recall that the agents' clocks could be asynchronized in phase. In order to avoid collisions, we enforce the agent to move in a "listen-think-walk" manner—namely, before moving from one waypoint wp_a to the other waypoint wp_b, the agent will wait at wp_a long enough, more than Δt amount of time to be specific, so as to collect the neighbor's information and broadcast its information to the neighbors. Moreover, when the agent waits at waypoint wp_a, it keeps using the messages received in last Δt amount of time to determine whether it is safe to move to the wp_b. The collision-free guarantees of this traffic scheduling strategy is shown in Section IV-A.

2) Goal Manager: In goal manager component, when agent receives a message from its neighbor, the agent will first check that if this neighbor are holding the same goal point as it does, if so, then the one whose current position is lexically smaller will change its goal (Algorithm 4, Lines 3–10). After this, the agent will then check whether the conditions (Algorithm 4, Lines 11, 16) for goal swap are triggered, if so, then it tries to execute the *2-way handshake* with the intended agent, and if the *2-way handshake* succeeds, the agents then updates their goals accordingly.

Note that this thread is executed concurrently with the main component (Algorithm 2), as a result, when agent's goal changes in this thread (Algorithm 4, Lines 6, 9, 14, 20), Algorithm 2 (Lines 11–14) may change the agent's *next _step*, therefore, in order to avoid the physical collision that is incurred by the concurrency, right after changing the goal in Algorithm 4, the agent will reset the timer for the safety checking (Algorithm 4, Lines 7, 10, 15, 21).

IV. Theoretical Results

A. Safety

In this section, we show that if the assumptions proposed in Section II-A are satisfied, then our algorithm is safe, i.e., the algorithm is collision-free.

Recall that as shown in Section III-A, to provide collision-free guarantee, the implementation of motion planner should satisfy Constraints III-A.1 and III-A.2. First, we show that our implementation satisfies Constraint III-A.1.

Lemma IV-A.1: Let $wp_{a_i}(t)$ be the waypoint that agent a_i is claiming at time t. If agent a_i changes the wp_{a_i} from $wp^0_{a_i}$ to $wp^1_{a_i}$ at time t^*, then there is no agent a_j such that $wp_{a_j}(t^*) = wp^1_{a_i}$.

Proof: Lemma IV-A.1 suggests that the algorithm can guarantee that when an agent changes the waypoint it is moving to, there is no other agent moving to this waypoint, or staying at this waypoint, at the same time. We prove Lemma IV-A.1 by contradiction. Let Δt be $\frac{2}{f_{\text{comm}}}$, according to the Algorithm 2, in order to allow agent a_i to change wp_{a_i} from $wp^0_{a_i}$ to $wp^1_{a_i}$ at time t^*, the following conditions must hold.

- *Condition 1:* For each message msg that a_j received between $t^* - \Delta t$ and t^*, msg should **not** trigger the two conditions stated in Algorithm 2, Lines 26, 28.
- *Condition 2:* For all the times $t \in (t^* - \Delta t, t^*]$, $wp_{a_i}(t) = wp^0_{a_i}$, as a_i needs to wait at $wp^0_{a_i}$ for more than Δt amount of time before changing wp_{a_i} (Algorithm 2, Line 30).

For any other agent $a_j \neq a_i$, $wp^1_{a_j}$ denotes the $wp_{a_j}(t^*)$ and $wp^0_{a_j}$ denotes the waypoint that a_j is claiming prior moving to $wp^1_{a_j}$, moreover, we use $t_j^{0\to1}$ to denote the time that a_j changes wp_{a_j} from $wp^0_{a_j}$ to $wp^1_{a_j}$. Suppose that there is an agent a_j such

that $wp^1_{a_j} = wp^1_{a_i}$. We here exhaustively outline two possible cases.

Case 1: $t_j^{0\to1} \leq t^* - 0.5\Delta t$: This case suggests that for all the times $t \in [t^* - 0.5\Delta t, t^*]$, $wp_{a_j}(t) = wp^1_{a_i}$. On the other hand, since a_j transmits the messages at fixed frequency f_{comm}, during time span $[t^* - 0.5\Delta t, t^*]$, it will transmit at least one message to the neighbors, as a result, a_i will receive a message that triggers Algorithm 2, Line 26 during $[t^* - 0.5\Delta t, t^*]$, which contradicts to Condition 1.

Case 2: $t^* - 0.5\Delta t < t_j^{0\to1} \leq t^*$: By Condition 2, in this case we have: For all the times $t \in (t^* - \Delta t, t^* - 0.5\Delta t]$, it holds that $wp_{a_j}(t) = wp^0_{a_j}$ and $wp_{a_i}(t) = wp^0_{a_i}$. Since we assume $wp^1_{a_i} = wp^1_{a_j}$, we can conclude that these two agents intend to go the the same waypoint, i.e., $wp^1_{a_i}$, during $(t^* - \Delta t, t^* - 0.5\Delta t]$. On the other hand, the period of time between $t^* - \Delta t$ and $t^* - 0.5\Delta t$ is long enough that a_i, a_j will have communicated *next_step* to each other. Given the fact that a_j changes wp_{a_j} after receiving a_i's *next_step*, one can conclude that a_j has higher priority to move to the $wp^1_{a_j}$, as a result, the message transmitted by a_j during $(t^* - \Delta t, t^* - 0.5\Delta t]$ will trigger Algorithm 2, Line 28 on a_i, which contradicts the Condition 1, completing the proof. ■

Theorem IV-A.1: At time $t = 0$, if each agent starts with a unique waypoint, then for any time $t > 0$, Constraint III-A.1 will be satisfied.

Proof: If each agent starts with a unique waypoint, then Lemma IV-A.1 suffices to show that Theorem IV-A.1 holds. ■

Next, we show that our implementation satisfied the Constraint III-A.2 as well.

Theorem IV-A.2: At time $t = 0$, if each agent starts with a unique waypoint, then for any time $t > 0$, Constraint III-A.2 will be satisfied.

Proof: We prove Theorem IV-A.2 by contradiction. Suppose that at time t^*, there are two agents a_i, a_j traveling on the edge connecting waypoint wp^0, wp^1 in the opposite direction. Without loss of the generality, we assume a_i is moving from wp^0 to wp^1 and a_j is moving from wp^1 to wp^0. Let $t_i^{0\to1}$ be the time that a_i changes wp_{a_i} from wp^0 to wp^1 and $t_j^{1\to0}$ be the time that a_j changes wp_{a_j} from wp^1 to wp^0, we have two cases.

Case 1: $t_i^{0\to1} \neq t_j^{1\to0}$: This case contradicts to Lemma IV-A.1, as the agent who changes its wp later will change its wp to a waypoint that is already claimed by the other.

Case 2: $t_i^{0\to1} = t_j^{1\to0}$: By Algorithm 2, Line 30, before $t_i^{0\to1}$, there is sufficient time for each of these two agent to sense that its *next_step* is claimed by the other, as a result, Algorithm 2, Line 26 will be triggered, and the agents' wp s will not change, where contradiction occurs. ■

B. Almost Sure Convergence

In this section, we show that if the *new goal selector* can pick a valid new goal point with nonzero probability, then the algorithm can enable the swarm to successfully form the desired shape with probability one, regardless of the swarm's initial configuration.

To prove the convergence of the algorithm, for every time step t, we construct the following objective functions:

$$J_1(t) = \sum_{i=1}^{|A|} d_i(t)$$

$$J_2(t) = |A| - \sum_{i=1}^{|Q|} e_i(t).$$

In which, $d_i(t)$ is the Manhattan distance from agent a_i's current position to its current goal at time *t*, i.e., the number of edges to be traversed in the grid. Moreover, for each goal position q_i, we define $e_i(t)$ as follows:

$$e_i(t) = \begin{cases} 1, & \text{if at time } t,\ \exists \text{ j s.t. } T_{a_j}(t) = q_i \\ 0, & \text{otherwise.} \end{cases}$$

One can see that these two objective functions will both equal zero only if all agents successfully arrive at a unique goal. Therefore, it is sufficient to show that our method can always drive the swarm to the desired final configuration by proving our method can make both J_1 and J_2 converge to zero, regardless of the initialization.

Proposition IV-B.1: Let $Pr\{\cdot\}$ be the probability that event $\cdot$ will occur, for any function $J(t) \in \mathbb{Z}^{\geq 0}$, it will almost surely converge to zero if:

- $\exists C \in \mathbb{Z}^{\geq 0}$, s.t. $\forall t$, $J(t) \leq C$;
- $\forall\, t_1 \leq t_2$, $J(t_1) \geq J(t_2)$;
- $\exists \tau, \epsilon > 0$, s.t. $\forall t$, $Pr\{J(t+\tau) \leq J(t) - 1 | J(t) \neq 0\} \geq \epsilon$

Proof: Proposition IV-B.1 suggests that: if a bounded non-negative objective is nonincreasing and will strictly decrease with a nonzero probability when it is not zero, then it will almost surely converge to zero. Without loss of generality, let $J(t_0) = C$, we have

$$\lim_{n\to\infty} Pr\{J(t_0 + n\tau) = 0\} \geq \lim_{n\to\infty} \sum_{k=C}^{n} \binom{n}{k} (\epsilon)^k (1-\epsilon)^{n-k}$$

Hence, $\lim_{n\to\infty} Pr\{J(t_0 + n\tau) = 0\} = 1$ ■

Next, to prove that both J_1 and J_2 can almost surely converge to zero, we show that both these two functions satisfy all three conditions proposed in Proposition IV-B.1.

Lemma IV-B.1: Both J_1 and J_2 are bounded by a finite constant.

Proof: At any time *t*, we can always find a minimal rectangle that covers all agent positions and goal positions. Let cx_t, cy_t be the length of the rectangle's edge in the *x*- and *y*-direction, respectively, and let c_t be $cx_t + cy_t$. Using the fact that each agent moves to its goal point greedily, we have $\forall t_1, t_2$, if $t_1 \leq t_2$, then $c_{t_1} \geq c_{t_2}$, in the other words $\forall t, c_t \leq c_{t0}$. Note that c_t is always larger than or equal to the Manhattan distance between any pair of points in the rectangle at time t, therefore, we have $J_1 \leq |A| c_{t0}$. On the other hand, since at least one goal will be assigned to the swarm, we have $J_2 \leq |A| - 1$, which completes the proof. ■

Lemma IV-B.2: J_2 is monotonically decreasing, moreover, J_1 is monotonically decreasing if J_2 equals zero.

Proof: Recall the way that the *new goal selector* works—if two agents realize that they are holding the same goal, then only one of them will select a new goal while the other agent will keep holding the current goal. As a result, no matter whether the new goal is valid or not, J_2 will never increase.

On the other hand, if $J_2 = 0$, then the *new goal selector* will no longer be triggered, so the robots will move greedily toward their goals. Thus, when $J_2 = 0$, J_1 will never increase. ■

To describe swarm's traffic condition, at every time step t, we construct a directed graph $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}_t)$, in which $\mathcal{V} = \mathrm{A}$ and $\mathcal{E}_t = \{(v_i, v_j)\}$ as its edge set, where $(v_i, v_j) \in \mathcal{E}_t$ if a_j occupies a_i's next waypoint at time t. By definition, each vertex on $\mathcal{G}_t$ essentially characterizes an agent, therefore, in the rest of the section we use the notation a_i and v_i interchangeably. For the sake of the description, at time t we call agent a_j is agent a_i's successor, or a_i is a_j's predecessor, if $(v_i, v_j) \in \mathcal{E}_t$. Additionally, we call those agent(s) a_i whose out degree $deg^+(a_i) = 0$ the *head* agent(s), in the other words, a *head* agent is an agent that is not blocked by any other agent.

Lemma IV-B.3: If $(v_i, v_j) \in \mathcal{E}_t$, then a goal swap between a_i and a_j will happen with a nonzero probability.

Proof: If $(v_i, v_j) \in \mathcal{E}_t$, then the goal swap between a_i and a_j will not increase $d_i + d_j$. By Algorithm 4 (Lines 16–21), the goal swap between a_i and a_j will happen with a nonzero probability, completing the proof. ■

Lemma IV-B.4: If $J_2 = 0$, then if $J_1 \neq 0$, J_1 will decrease by at least 1 within a finite amount of time with nonzero probability, independent of the history.

Proof: We prove Lemma IV-B.4 via case analysis. If $J_1(t) \neq 0$, then at least one agent intends to move to the next waypoint, there are three possible cases.

- *Case 1:* The graph $\mathcal{G}_t$ is cyclic:

Assume that at time t^0, there exists a cycle $\mathcal{C}$ on $\mathcal{G}_{t^0}$. Let $len(\mathcal{C})$ be the length of $\mathcal{C}$, i.e., the number of the agents that are on cycle $\mathcal{C}$ at time t^0. We use $a^1, a^2, \ldots, a^{len(\mathcal{C})}$ to denote the agents that are on $\mathcal{C}$ at time t^0, for the sake of description, we order these agents in the way such that:

- if $i < len(\mathcal{C})$, then $(a^i, a^{i+1}) \in \mathcal{E}_{t^0}$;
- if $i = len(\mathcal{C})$, then $(a^i, a^1) \in \mathcal{E}_{t^0}$.

Note that it does not matter how we pick the first agent a^1, a^1 could be any agent that is on cycle at time t^0.

We show that Lemma IV-B.4 holds in Case 1 by contradiction.

Suppose that $\forall \tau > 0, Pr\{J_1(t_0 + \tau) \leq J_1(t_0) - 1\} = 0$, i.e., the probability that J_1 strictly decreases after t_0 is 0. One can conclude the following if the assumption is true:

- no agent can change its position after t_0, as J_1 will decrease every time when any agent moves, moreover;
- no agent can execute the goal swaps that can decrease J_1.

In other words, the only two possible actions left for agents are—doing nothing, or executing the goal swaps that cannot change J_1 (Algorithm 4, Lines 16–21). It is worth noting that if these two actions are the only ones available for agents, then each time when an agent tries to decide an action to execute, the action "doing nothing" will be picked with a nonzero probability, as stated in Algorithm 4 (Lines 16–21).

Next, let q^* be a^1's goal at t^0, combing Lemma IV-B.3 and the conclusion we just obtained, we have that the following will happen with a nonzero probability.

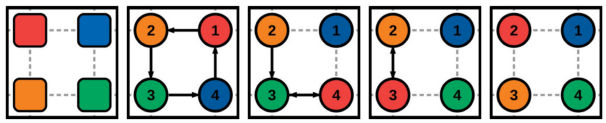


Fig. 5. From left to right: Shape that contains four goal points; a sequence of events where J_1 strictly decreases. The frames of this sequence of events are ordered from left to right. All the information is encoded in the same way as Fig. 2.

Fig. 6. Illustration of two possible cases (shown by a sequence of two figures on the left and a sequence of two figures on the right) where the *head* agent has already arrived at its goal. The goal shape is shown in Fig. 5 (left). All the information is encoded in the same way as Fig. 2.

- q^* propagates among the agents $a^1, \ldots, a^{len(\mathcal{C})}$ via **swap** in the order that: $a^1 \to a^{len(\mathcal{C})} \to a^{len(\mathcal{C})-1} \to \cdots \to a^2$.
- For any agent $a^i \neq a^1$ that is on $\mathcal{C}$ at time t^0, it will **do nothing** but wait to take q^* from its successor (via goal swap) and then pass q^* to its predecessor (via goal swap).

That is, q^* will traverse agents $a^1, \ldots, a^{len(\mathcal{C})}$ in the opposite direction of $\mathcal{C}$ with a nonzero probability, a graphical illustration of this event is shown in Fig. 5.

Let t^1 be the time that q^* is passed to a^2, we have $J_2(t^1) \leq J_2(t^0) - len(\mathcal{C})$, because the owners of goals of all the agents that are on $\mathcal{C}$ have moved one step closer to goal via goal swap (see Fig. 5 for a more intuitive illustration). This observation suffices to show that it is incorrect to assume that the probability that J_1 strictly decreases after t_0 is 0, completing the proof in Case 1.

On the other hand, if the $\mathcal{G}_t$ is acyclic, we can find at least one *head* agent, i.e., the agent that is not blocked by any other agent. The *head* agent(s) can either be an agent that already arrived at the goal, or an agent that is still moving to the goal.

- *Case 2:* The graph $\mathcal{G}_t$ is acyclic, moreover, there exists one head agent a_i that has not arrived at its goal $T_{a_i}(t)$ yet:

If at least one *head* agent has not arrived at its goal yet, J_1 will decrease by one because its next waypoint is open, thus, Lemma IV-B.4 holds in case 2.

- *Case 3:* The graph $\mathcal{G}_t$ is acyclic, moreover, all the head agents have arrived at their goals:

If all the *head* agents have already arrived at their goals, then there is at least one non *head* agent who is blocked by a *head* agent. Using Lemma IV-B.3, we have that the blocked agent can make progress with a nonzero probability by swapping with the blocking *head* agent that has already arrived at its goal. With at most $|A| - 1$ swaps, the agents will either form a cycle, as shown in Fig. 6 (right), which is Case 1, or generate a *head* agent that has not reached its goal, shown in Fig. 6 (left), which is Case 2. Hence in Case 3, J_1 will decrease within a finite amount of time with nonzero probability as well, completing the proof. ■

Lemma IV-B.5: The *motion planner* will make the position of each assigned goal's owner to greedily move toward the position

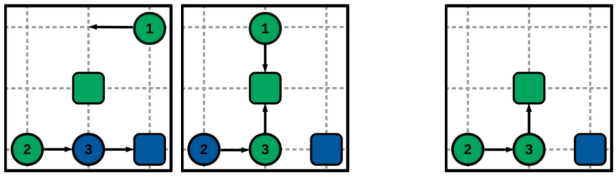


Fig. 7. Illustration of two possible cases (shown by a sequence of two figures on the left and a single figure on the right) where two of q_d's owners (circles in green) meet each other. All the information is encoded in the same way as Fig. 2.

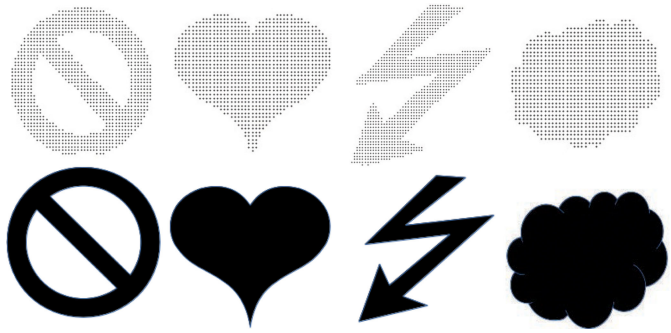


Fig. 8. 1024 agents form four user-defined shapes. (Bottom) example input binary images and (top) corresponding collective formations.

of the goal point with nonzero probability, regardless of the swarm's configuration.

Proof: We prove this lemma via case analysis. At time t, for any agent a, let q be $T_a(t)$, i.e., agent a's target at time t. We here exhaustively outline all two possible cases.

- *Case 1:* a has not arrived at q:

If a has not arrived at q, a will intend to move to the next waypoint to get closer to q. If the waypoint is unoccupied, then a moves one step closer to q. Otherwise, if the waypoint is occupied, using Lemma IV-B.3, q will be passed to the agent that is occupying the waypoint with nonzero probability. In either of these two cases, the distance between position of q's owner and q decreases by one with nonzero probability.

- *Case 2:* a is already at q:

If a is already at q, then a can stay at q with nonzero probability, completing the proof. ■

Lemma IV-B.6: If $J_2 \neq 0$, then the event that: two agents that are holding the same goal at the same time are located within distance R, will occur within a finite amount of time with nonzero probability. Namely, at time $\bar{t}$, if $J_2 \neq 0$, then $\exists\, \overline{\tau}, \bar{\epsilon} > 0, a_i \neq a_j$, such that

$$Pr\{||p_{a_i}(\bar{t}+\overline{\tau}) - p_{a_j}(\bar{t}+\overline{\tau})|| \leq R, T_{ai}(\bar{t}+\overline{\tau}) = T_{aj}(\bar{t}+\overline{\tau})\} \geq \bar{\epsilon}$$

Proof: If $J_2 \neq 0$, then at least two agents are holding the same goal point. Let q_d be a goal which is assigned to more than one agent. Lemma IV-B.5 is sufficient to show that two of q_d's owners can concurrently keep moving greedily to q_d with nonzero probability, shown in Fig. 7 (left), unless one of q_d's owners blocks the other owner's way in the trip, shown in Fig. 7 (right). However, one agent being blocked by the other implies that the distance between two agents is one grid length, which is smaller than R, completing the proof. ■

Lemma IV-B.7: If $J_2 \neq 0$, then J_2 will decrease by at least one within a finite amount of time with nonzero probability, independent of the history.

Proof: Lemma IV-B.6 suggests that if $J_2 \neq 0$, then two agents holding the same goal will meet each other within a finite amount of time with nonzero probability, which means *new goal selector* will be triggered within a finite amount of time with nonzero probability. Additionally, when *new goal selector* (Algorithm 4, Lines 3–10) is triggered, all the goal points in Q will be picked with a nonzero probability, which implies that a goal that has not been assigned to swarm yet will be picked with nonzero probability. As a result, J_2 will decrease with nonzero probability, completing the proof. ■

Theorem IV-B.1: Both J_1 and J_2 will almost surely converge to zero, regardless of the swarm's initial configuration.

Proof: Using Lemmas IV-B.1, IV-B.2, IV-B.4, and IV-B.7, we can show that both J_1 and J_2 satisfy all three conditions proposed in Proposition IV-B.1. Hence, both J_1 and J_2 will almost surely converge to zero, regardless of the initialization of the swarm. ■

C. Complexity

In this section, we study the cost of implementation of the algorithm proposed in Section III-C with respect to its time complexity, memory complexity, and communication complexity.

First, we study the time complexity for each agent planning their action, i.e., the time complexity for executing Algorithm 2, Lines 9–35. One can see that the time cost is dominated by the time complexity for looping through all the messages received in last $\frac{2}{f_{\text{comm}}}$ amount of time. In the *msg_buff*, there will be at most $2|N_{a_i}(t)|$ amount of the messages, on the other hand, each agent can have at most $\lfloor \frac{2R}{l} \rfloor^2$ amount of neighbors in communication range, where R is agent's communication range, l is the grid length. This suffices to show that the time complexity for the decision making is $\mathcal{O}(\lfloor \frac{R}{l} \rfloor^2)$.

Next, the algorithm's memory footprint is dominated by the memory to store the input point set, as a result, the algorithm's memory complexity is $\mathcal{O}(|Q|)$.

To investigate the algorithm's communication complexity, i.e., the amount of data each agent will transmit during one unit of time, we first study the amount of data that each agent will transmit within each communication round, i.e., $\frac{1}{f_{\text{comm}}}$ amount of time. During each communication round, an agent a_i will broadcast one message with a length of $\mathcal{O}(1)$ (Algorithm 3, Line 2), and process at most $|N_{a_i}(t)|$ *2-way handshake*s, as a_i can receives no more than $|N_{a_i}(t)|$ amount of messages from the neighbors in communication range. Additionally, the amount of data exchanged to process a *2-way handshake* is $\mathcal{O}(1)$, as a result, during one communication round, the amount of data that each agent a_i will transmit to its neighbors is $\mathcal{O}(\lfloor \frac{R}{l} \rfloor^2)$, as each agent can receive no more than $\lfloor \frac{2R}{l} \rfloor^2$ neighbors in communication range, where R is agent's communication range, l is the grid length. On the other hand, in a unit of time, there will be $\mathcal{O}(f_{\text{comm}})$ communication rounds, which suggests that

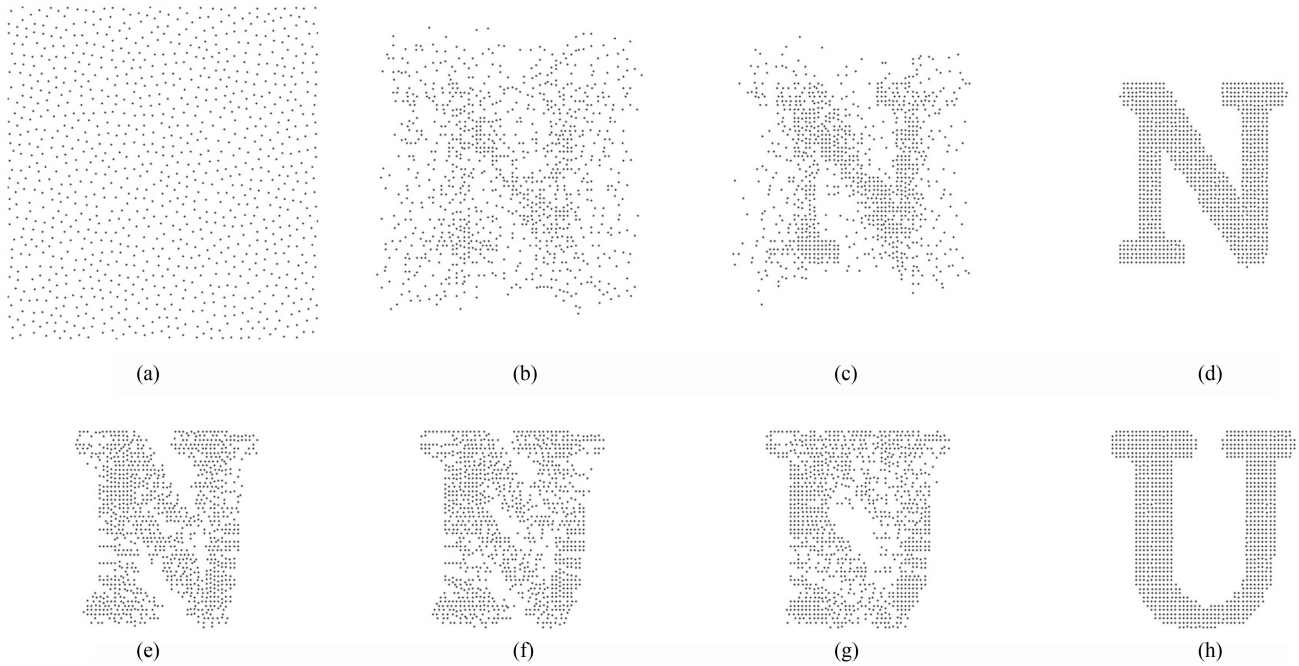


Fig. 9. Still images from simulation where 1024 agents try to form two different shapes in a row. In this simulation, a swarm of 1024 agents first form a letter "N," then switch to a letter "U" when it is detected that all robots have reached a goal. (a) T = 0 s. (b) T = 99 s. (c) T = 147 s. (d) T = 838 s. (e) T = 877 s. (f) T = 888 s. (g) T = 906 s. (h) T = 1031 s.

for each agent, the amount of data transmitted during a unit of time is $\mathcal{O}(\lfloor \frac{R}{l} \rfloor^2 f_{\text{comm}})$.

V. Performance Evaluation

To demonstrate the correctness and performance of the algorithm presented in this article, we implemented and tested it in both simulated and physical experiments. For all experimental tests, the shape was successfully formed. We also compared the performance of our proposed algorithm with the centralized algorithm proposed in [12]. In this centralized approach, every agent is initially assigned a unique goal. In addition, this assignment minimizes the total traveling distance. It is shown in [12] that with such optimal initial assignment, the agents' paths will form a acyclic direct graph , and each agent's motion can be then scheduled via vertex ordering. While the centralized method can produce the distance-optimal solution, which outperforms our method, it could suffer from a single-point of failure and therefore is less robust than our method.

A. Simulation

In simulation, each agent is modeled as an omnidirectional robot able to sense its position and orientation in a common coordinate system. We treat each agent as a circle with a radius of 0.05 m. Agents are able to communicate with any other agent who lies within its communication range of 0.6 m, and travel at a speed of 0.05 m/s. These values match the physical robot described in Section V-B1.

In each test, the goal shape is given to the swarm in the form of a binary figure, i.e., a black and white image. The figure is scaled such that the number of goal pixels on the figure equals the number of agents. Example input images and corresponding shapes formed by the collective are shown in Fig. 8. See Fig. 9 for images from one simulation where 1024 agents formed the letters "N" and "U" in sequence.

First, the simulation is used to investigate the effect of the swarm size on the algorithm's convergence time, i.e., the total time it takes for the swarm to complete shape formation as well as average robot travel distance, i.e., the total distance traveled by all robots normalized by the number of robots. In this task, swarms of size 16–529 agents formed a given target configuration from a random initialization. For every swarm size, 200 trials were run, and in each trial the target shape is randomly generated as a set of connected random positions. The large number of trials are able to eliminate the bias on the final result that is introduced by the target shape, since in each trial the target shape is randomly generated. Fig. 10 shows the distributions of convergence time as well as the average distance traveled for different swarm sizes, and a comparison to the centralized method [12].

One counter-intuitive observation here is that the convergence time for centralized methods does not monotonically increase over the swarm size, sometimes even goes down. It is shown in [12] that the worst case convergence time for the centralized method is $|A| + d_{\max} - 1$ where $|A|$ is the number of agents, and $d_{\max}$ is the maximal individual travel distance (the distance from one agent's initial position to its goal) among swarm. On the other hand, in simulations, the swarm moved in a fixed-size arena, hence when swarm size $|A|$ increases, i.e., the density of the swarm increases, the $d_{\max}$ will decrease. As a result, the convergence time for centralized methods will not necessarily increase over the swarm size. The fact that agents move in a fixed-size arena can also help to explain the trend of the total

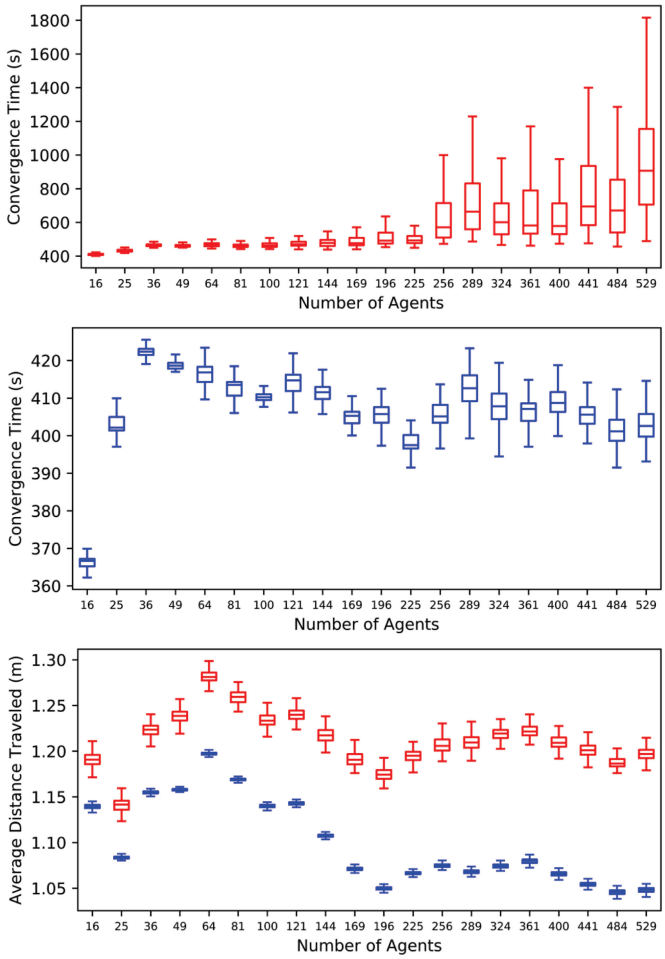


Fig. 10. Simulation results for both our method (red) and the centralized method [12] (blue) in standard box-plot format. For each number of agents, 200 trials were run, and in each trial the target shape was randomly generated.

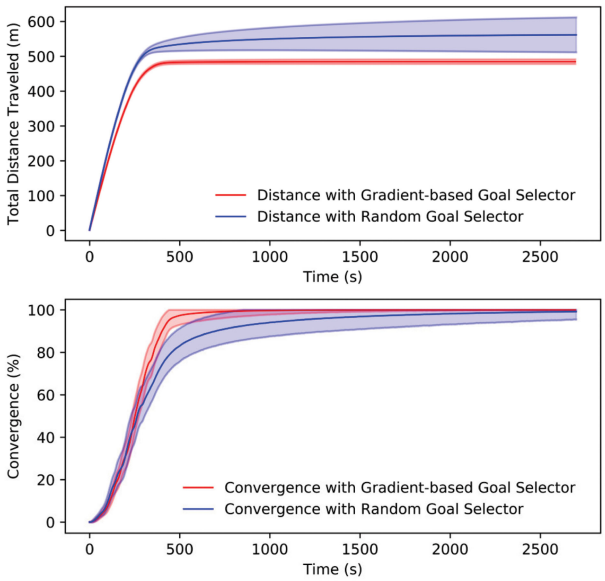


Fig. 11. Illustration of the improvement made by the gradient-based selector on the algorithm convergence rate and total distance traveled compared to using a random selector. Each solid line in the plot is the average result from 40 000 simulations of 400 agents, and the colored shade areas show the confidence interval for convergence and total distance traveled over time at a confidence level of 2σ (two standard deviations above or below the average).

distance plots. As $|A|$ increases, the swarm's initial position will be "closer" to the target positions (one can consider an extreme case where number of the agents equals to the number of the vertices in arena, in this case, the average distance traveled will be zero), hence the overall trend of the distance plot is that the average distance traveled goes down as number of agents goes up. In these plots, we can see that the distance traveled incurred by our method is only around 20% more than the one that is incurred by the centralized method. Moreover, when the density of the swarm is low (less than 225 agents in arena), the difference on convergence times for both methods are considerably small. When the density of the swarm increases, the convergence time for our method sharply increases, this is because in this case, the random goal swaps, i.e., the goal swaps to resolve the deadlock, will more likely happen, as a result, the algorithm will converge slower.

A second test compares the two approaches for *new goal selector*. It measures the convergence rate as well as the total distance traveled by a fixed-size swarm. In this experiment, 400 agents try to form 200 different randomly generated shapes. For each shape, the swarm executes the formation algorithm 200 times, giving a total of 40 000 simulation runs. For each of these runs, agents are initialized with a uniform random distribution centered at the shape's center of mass. For every time step, we measure the average completion rate, the average distance traveled, and the confidence interval at a confidence level of 2σ for both convergence and distance travel, at that time for all 40 000 runs. The results are shown in Fig. 11. In these plots, we can see that the gradient-based selector can dramatically increase the algorithm's convergence rate and helps to eliminate the long tail of convergence incurred by the random goal selector. Besides that, unsurprisingly, the gradient-based goal selector can also reduce the variance of the convergence time and total distance traveled.

Simulation was also used to compare the convergence rate as well as swarm's total traveled distance for both our method and the centralized one. The experiment contains 40 000 trials; in every trial, 400 agents tried to drive toward a set of randomly generated goal points. First, agents were initialized with random starting locations. Next, either our method or the centralized method was used to drive the agents to the goal points. The simulation results of our method and the centralized method are shown in the Fig. 12. The difference between Figs. 10 and 12 is that Fig. 10 shows the statistics of the final solution's quality, i.e., the total distance and convergence time, whereas Fig. 12 helps to understand how the algorithm's convergence and distance traveled change during execution. The plot shows average result and the confidence interval with a confidence level of 2σ for both the convergence rates and total distance traveled for both methods. Note that at the beginning, from time 0 to 80 s, our algorithm makes faster progress than the centralized method. This is because in the centralized method, agents take goals located in the inner area of the shape first so they will not block

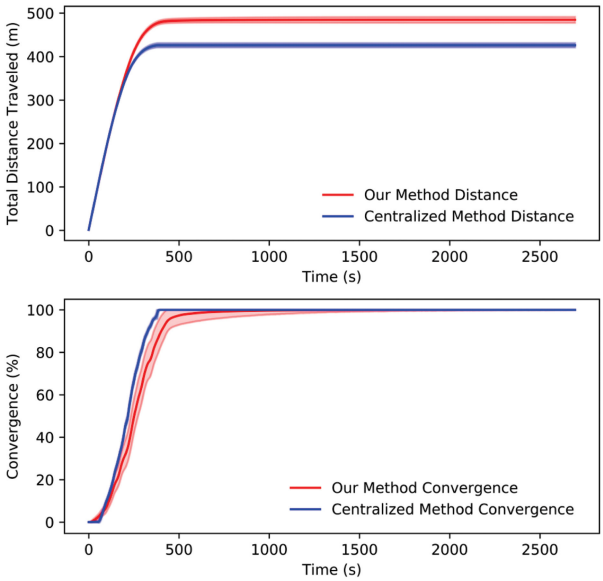


Fig. 12. Performance comparison between our method (red line) and centralized method (blue line). Each solid line in the plot is the average result from 40 000 simulations of 400 agents, and the colored shade areas show the confidence interval for convergence and total distance traveled over time at a confidence level of 2σ (two standard deviations above or below the average).

other moving agents' paths, while in our method, agents initially choose goals at random. As a result, the agents are more likely to take the goal points nearby first, giving our method a short-term win at the beginning.

B. Experiments

To validate the correctness and efficiency of our algorithm beyond simulation, we performed several physical experiments using the *Coachbot V2.0* swarm system, a custom-made swarm of 100 differential-drive wheeled robots as shown in Fig. 13.

1) Hardware: There are two key components in the swarm's hardware design—the robot itself, and the base station used to manage and operate the swarm.

The design of *Coachbot V2.0* consists of four key components—a main computer, localization module, power module, and locomotion module. The main computer is a Raspberry Pi 3b+ with a 1.4-GHz 64-bit quad-core CPU, 1-GB RAM, and dual-band 802.11ac wireless LAN (2.4 and 5 GHz). The localization module, shown in Fig. 13, is a custom PCB consisting of two TS3633-CM1 sensors and an Atmel attiny87 microprocessor. The two sensors receive infrared signals from the ceiling mounted *HTC vive* lighthouse which emits a time-varying infrared pattern, allowing the sensors to determine their position. The microprocessor calculates the positions of the two TS3633-CM1 sensors at an update rate of 30 Hz, and sends the sensors' positions to the Raspberry Pi via the UART. With this information, the main board can calculate the robot's orientation and position accurate up to 15° and 0.04 m. The robot is powered by a 2.5 Ah lithium-ion rechargeable battery. Additionally, a Bluetooth low energy module can disconnect the battery power from the rest of the robot. This allows a user to send a Bluetooth command to put robots into a low-energy sleep mode as well as wake robots up from this sleep mode. With a fully charged battery, robots can operate for approximately four hours, or sleep for three months. The robot drives across a flat surface using two wheels driven by dc motors. An H-bridge controls the speed and direction of each motor independently, allowing for differential drive. Each robot has a height of 0.12 m and a radius of 0.05 m.

A computer workstation manages all robots in the swarm at once, allowing a single person to easily operate the entire swarm without any direct interaction. However, the workstation does not control robots during experiments. The workstation can communicate with all robots using Wi-Fi, and can power the robots on and off using Bluetooth. Custom-made software on the workstation uses these capabilities to monitor the status of all robots, update the code executed on the swarm, start/stop the robots' program, turn them ON/OFF, and collect the data from experiments.

2) Software: Software running in a modified version of Raspbian operating system is running onboard each robot's Raspberry Pi, controlling the robot's basic behaviors, robot-to-robot communication, and interaction with the base station.

There are two separate communication channels in the system—a star network between base station and robots, and a mesh network among the robots. These two communication channels serve two separate tasks [see Fig. 13 (2)]. Both channels make use of the Wi-Fi capabilities of the onboard Raspberry Pi. The star network is mainly used for transferring files between the base station and robots, i.e., uploading the user programs to the robots, while the other channel will support robot-to-robot communication used for the user programs. The star network is implemented by connecting the robots to the base station's Wi-Fi router. Communication along this channel uses standard TCP/IP protocol to ensure the reliable transfer of data and files. For the robot-to-robot channel, communication uses layer 2 broadcasting, i.e., MAC address-based broadcasting. This allows robots to communicate with other robots directly, without use of the base station router. By embedding the robot's position in the data packet, the robots' communication range in experiments can be artificially limited to a desired value by actively discarding messages which originate outside the desired communication range. In this article's experiments, the robots' communication range is limited to 0.6 m.

A custom coordination system makes use of the communication on the star network to operate and manage the swarm in an easy, scalable way [53], [54]. This system has three software components—an FTP server, a monitor module, and a broadcaster module. The FTP server is used to update the code running on the robots. It can push new software to all robots at once. The monitor module is used to monitor the status of all robots, which is transmitted from each robot to the base station. It monitors information, such as battery voltage, firmware version, etc., and displays it to the operator. The broadcaster module is used to send commands to the swarm which help in its operation, such as starting and stopping execution of the user code, turning the robots ON/OFF, and moving robots to a charging station. The broadcaster module makes use of both Wi-Fi and Bluetooth communication.

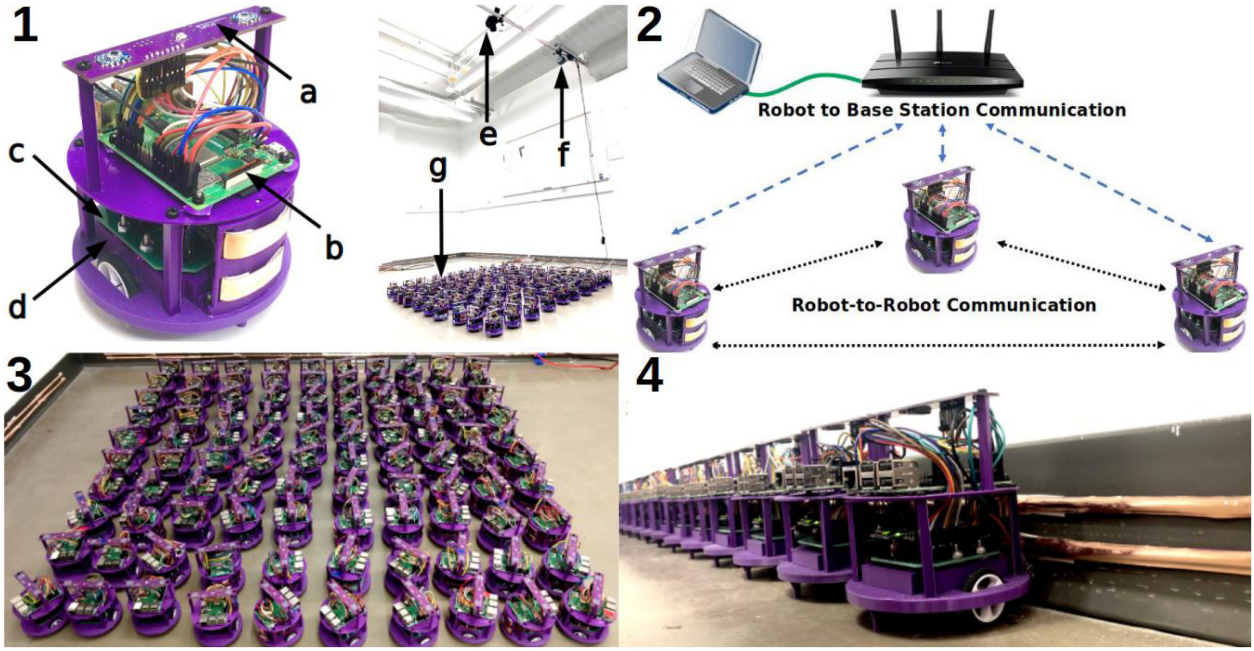


Fig. 13. Illustration of hardware used in experiments. (1) Key components of the *Coachbot V2.0* swarm system: (1 Left) Robot used in the experiments. The robot is in a cylinder shape with a height of 0.12 m and a radius of 0.05 m. Key components are: (a) Localization system based on the HTC Vive. (b) Raspberry PI b+ computer, (c) electronics mother board, and (d) rechargeable battery. (1 Right) Robot arena used in experiments: (e) overhead camera (only used for recording videos), (f) overhead HTC Vive base station, and (g) swarm of 100 robots. (2) Illustration of the *Coachbot V2.0* swarm communication network. The green link is an ethernet connection between the base station and the Wi-Fi router. The blue links are TCP/IP connections, and the black links are layer 2 broadcasting connections. (3) Swarm of 100 robots. (4) Robots charging by connecting to two metal strips attached to the wall.

3) Dealing With Real-World Nonidealities: In reality, some assumptions proposed in Section II-A are difficult to be guaranteed in real robot hardware. To compensate the real-world nonidealities, such as communication errors, imperfect robot motion, sensing errors, etc., we relax the assumptions proposed in Section II-A as follows.

a) For any agent, the frequency of its clock is bounded, specifically: $\exists f^{\max}_{\text{clock}}, f^{\min}_{\text{clock}}$ s.t. for any agent a_i, we have $f^{\min}_{\text{clock}} \leq f^{i}_{\text{clock}} \leq f^{\max}_{\text{clock}}$.
b) Note that for each agent a_i, its communication rate f^{i}_{comm} is defined according to its onboard clock, i.e., the clock that is with frequency f^{i}_{clock}. As a result, even though each agent is programmed to broadcast at the same frequency f_{comm} (Algorithm 3, Line 4), from a global observer's perspective, their communication rate could be still different due to the difference on clock's frequency f^{i}_{clock}.
c) The interagent communication packet loss rate is small enough such that: for each robot, if it sends the same message m times in a row, it is guaranteed that this message can be received by all its neighbors in communication range.
d) For any agent a_i, the speed v^{i}_{m} at which it moves on a grid edge is bounded—namely, $\exists v_{\max}, v_{\min}$ s.t. for any agent a_i, we have $v_{\min} \leq v^{i}_{m} \leq v_{\max}$.

First, note that as shown in Section IV-A, the proof of Theorems IV-A.1 and IV-A.2 does not rely on any assumption about robot's physical speed, in other words, only the relaxation of assumptions (a), (b), and (c) will effect the correctness of Theorems IV-A.1 and IV-A.2. To preserve the correctness of Theorems IV-A.1 and IV-A.2, in practice, we extend Δt to make the robot to act more conservatively, so as to compensate the difference on robots' clock frequency and packet loss. To be specific, we extend Δt such that

$$\Delta t \geq \frac{2\,m}{f_{\text{comm}}} \frac{f^{\max}_{\text{clock}}}{f^{\min}_{\text{clock}}}.$$

Here, the first term $\frac{2\,m}{f^{\min}_{\text{comm}}}$ is for accommodating the communication loss, and the second term $\frac{f^{\max}_{\text{clock}}}{f^{\min}_{\text{clock}}}$ is for compensating the difference on robots' clock frequency. The reason for the second term is that when one agent executes Algorithm 2, Line 30 and Algorithm 3, Line 4, it will use the onboard clock to do the calculation, and different clock frequency will yield different results. Therefore, we add the second term to guarantee that for the robot with the fastest clock, it will wait long enough to accommodate the one with the slowest clock.

So far, we showed that the correctness of Theorems IV-A.1 and IV-A.2 can be preserved when the assumptions proposed in Section II-A are relaxed. However, when agents move on grids in different speeds, collisions could still happen. To accommodate the difference on robot's physical speeds, we stretch the grid length l so as to give robots some "buffer space." Specifically, we want l large enough such that

- When two robots move on two orthogonal adjacent edges, no collision happens, i.e.,

$$\min \sqrt{(l - v_{\max} t)^2 + (v_{\min} t)^2} \geq 2r, \text{ subject to } t \in \left[0, \frac{l}{v_{\max}}\right].$$

- When two robots move on two collinear adjacent edges, no collision happens, i.e.,

$$\min(l - v_{\max}t + v_{\min}t) \geq 2r, \text{ subject to } t \in \left[0, \frac{l}{v_{\max}}\right].$$

Solving those two inequalities above, we have

$$l \geq 2\frac{\sqrt{v_{\max}^2 + v_{\min}^2}}{v_{\min}}r.$$

Additionally, in the algorithm, we have the assumption that the robot can move in any direction directly, which does not hold for the *Coachbot V2.0*, as *Coachbot V2.0* is a differential drive robot. When a *Coachbot V2.0* moves from one waypoint wp_a to another waypoint wp_b, it will first spin at waypoint wp_a to adjust its orientation to be parallel with the grid edge connecting wp_a and wp_b, before moving to wp_b. Note that for each step, the robot may change its orientation by 0 or $\frac{\pi}{2}$ *rads* (because the robot can move both forward and backward, hence it does not need to adjust its orientation by more than $\frac{\pi}{2}$ *rads*). Different heading adjustments will take different amount of time, as a result, for those robots that transit to their next waypoints at the same time, the robots who need to spin by $\frac{\pi}{2}$ *rads* will start moving toward their next waypoint later than the ones that do not need to spin, so a collision may occur. To compensate this difference on the adjustments of the robot's heading, we introduce another type of "buffer space" to grid length l, that is to say, assume the robot's minimal spin speed is ω^*, we enforce the grid length l to be

$$l \geq 2\frac{\sqrt{v_{\max}^2 + v_{\min}^2}}{v_{\min}}r + v_{\max}\frac{\pi}{2\omega^*}.$$

In experiments, our choice of grid length l is 0.20 m.

4) Results: In these physical experiments, we demonstrate that our algorithm can be easily implemented on a relatively large scale physical swarm, and it can provide reliable performance. Additionally it is robust to real-world noise in both communication, sensing, and motion. In this experiment, 100 robots start randomly dispersed and form the letters "N," "U" in sequence. With the help of hop-count information, robots can detect when the first letter is completed and then switch to form the second shape, an "U." Images from one of these experiments using our algorithm is shown in Fig. 1. We also compared the real-world performance between our algorithm and the centralized approach. A shape was formed 15 times with both approaches, and we compared the average convergence rate and average total distance traveled for both approaches. In all these 30 experiments, the shape formation successfully completed. The results from this comparison experiment are shown in Fig. 14.

In these plots, we can observe that our method gets a short-term win of convergence rate at the beginning compared to the centralized method, which is consistent with the simulation result. On the other hand, one observation here is that in the simulation plots (see Fig. 12), both the convergence and the distance traveled monotonically increase over time, whereas in Fig. 14, during the first 20 s, the convergence plot fluctuates. This is because, in simulation, the agents are tasked to form a set of random shapes from a random initialization of positions whereas

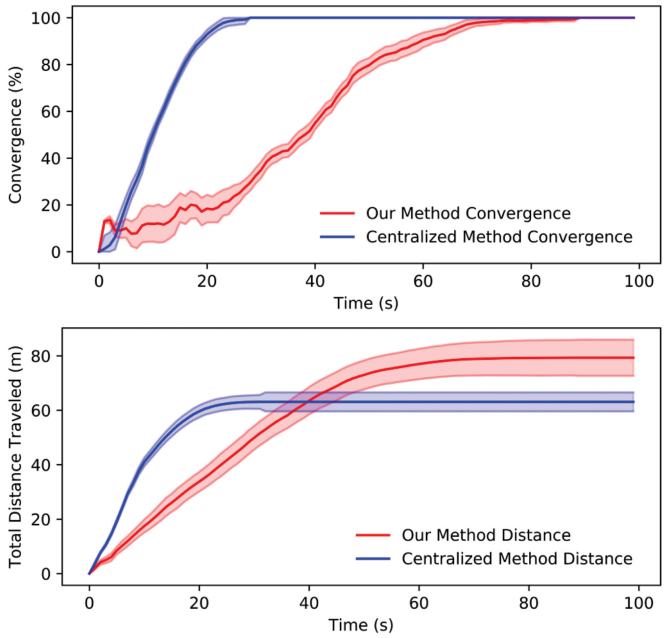


Fig. 14. Illustration of average performance comparison between our method (red line) and the centralized method (blue line). Each solid line is the average result from 15 physical experiments of 100 robots, and the colored shade areas show the confidence interval for convergence and total distance traveled over time at a confidence level of 2σ (two standard deviations above or below the average).

in all physical experiments the robots are tasked to form the same shape, letter "N," which will introduce the bias to the final result. Additionally, noise in robot's motion, communication, and sensing, cannot be captured by the simulation very well, and the number of trials are not large enough to eliminate the noise's effect on the convergence rate. As a result, the convergence plot for physical experiment is not monotonically increasing over time.

VI. Conclusion

In this article, we introduced a fully distributed shape formation algorithm which enabled a swarm of robots to move and form a user-specified shape quickly and without collision. With this algorithm, agents took an array of goal points as input, then used the local information to distribute the goals among the swarm and schedule the collision-free paths concurrently. To demonstrate the correctness and performance of our algorithm, we executed our algorithm on a swarm of up to 1024 simulated robots and 100 real robots. The result of these experiments showed that while this algorithm was slower than the centralized approach, it could still reliably converge to all robots forming the desired shape. Additionally, the large numbers of simulation trials as well the real robot experiments showed that it could do so with only a small difference in travel distance, around 25% to be specific, when compared to an optimal centralized approach. For future work, we plan on addressing some of the current limitations of this work, such as removing the requirement of grid representation of the space, and using a coordinate system

generated using robot-to-robot communication and sensing, instead of relying on a global positioning system.

REFERENCES

[1] C.-H. Yu and R. Nagpal, "Sensing-based shape formation on modular multi-robot systems: A theoretical study," in *Proc. 7th Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2008, vol. 1, pp. 71–78.

[2] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Mag.*, vol. 29, no. 1, pp. 9–20, 2008.

[3] J. Alonso-Mora, A. Breitenmoser, M. Rufli, R. Siegwart, and P. Beardsley, "Image and animation display with multiple mobile robots," *Int. J. Robot. Res.*, vol. 31, no. 6, pp. 753–773, 2012.

[4] S. W. Feng, S. D. Han, K. Gao, and J. Yu, "Efficient algorithms for optimal perimeter guarding," in *Proc. Robotics: Sci. Syst.*, Jun. 2019, doi: 10.15607/RSS.2019.XV.002.

[5] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Res. Logistics Quart.*, vol. 2, no. 1/2, pp. 83–97, 1955.

[6] D. P. Bertsekas and D. A. Castañon, "Parallel synchronous and asynchronous implementations of the auction algorithm," *Parallel Comput.*, vol. 17, no. 6/7, pp. 707–732, 1991.

[7] R. M. de Mendonça, N. Nedjah, and L. de Macedo Mourelle, "Efficient distributed algorithm of dynamic task assignment for swarm robotics," *Neurocomputing*, vol. 172, pp. 345–355, 2016.

[8] D. P. Bertsekas, "A new algorithm for the assignment problem," *Math. Program.*, vol. 21, no. 1, pp. 152–171, 1981.

[9] M. S. Hung, "A polynomial simplex method for the assignment problem," *Oper. Res.*, vol. 31, no. 3, pp. 595–600, 1983.

[10] M. Ji, S.-i. Azuma, and M. B. Egerstedt, "Role-assignment in multi-agent coordination," *Int. J. Assistive Robotics Mechatronics*, vol. 7, pp. 32–40, Jan. 2006.

[11] M. Turpin, N. Michael, and V. Kumar, "Capt: Concurrent assignment and planning of trajectories for multiple robots," *Int. J. Robot. Res.*, vol. 33, no. 1, pp. 98–112, 2014.

[12] J. Yu and S. M. LaValle, "Shortest path set induced vertex ordering and its application to distributed distance optimal formation path planning and control on graphs," in *Proc. 52nd IEEE Conf. Decis. Control*, 2013, pp. 2775–2780.

[13] M. Turpin, K. Mohta, N. Michael, and V. Kumar, "Goal assignment and trajectory planning for large teams of aerial robots," in *Proceedings of Robotics: Science and Systems*, Berlin, Germany Jun. 2013, doi: 10.15607/RSS.2013.IX.030.

[14] P. MacAlpine, E. Price, and P. Stone, "SCRAM: Scalable collision-avoiding role assignment with minimal-makespan for formational positioning," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 2096–2102.

[15] S. Li *et al.*, "Particle robotics based on statistical mechanics of loosely coupled components," *Nature*, vol. 567, no. 7748, pp. 361–365, 2019.

[16] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas, "Distributed multi-robot task assignment and formation control," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2008, pp. 128–133.

[17] S. L. Smith and F. Bullo, "Target assignment for robotic networks: Asymptotic performance under limited communication," in *Proc. IEEE Amer. Control Conf.*, 2007, pp. 1155–1160.

[18] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424–440, 2015.

[19] M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*, vol. 33. Princeton, NJ, USA: Princeton Univ. Press, 2010.

[20] M. Cao, C. Yu, and B. D. Anderson, "Formation control using range-only measurements," *Automatica*, vol. 47, no. 4, pp. 776–781, 2011.

[21] D. V. Dimarogonas and K. H. Johansson, "Further results on the stability of distance-based multi-robot formations," in *Proc. IEEE Amer. Control Conf.*, 2009, pp. 2972–2977.

[22] R. Olfati-Saber and R. M. Murray, "Graph rigidity and distributed formation stabilization of multi-vehicle systems," in *Proc. 41st IEEE Conf. Decis. Control*, Dec. 2002, vol. 3, pp. 2965–2971.

[23] K.-K. Oh and H.-S. Ahn, "Formation control of mobile agents based on inter-agent distance dynamics," *Automatica*, vol. 47, no. 10, pp. 2306–2312, 2011.

[24] J. Guo, Z. Lin, M. Cao, and G. Yan, "Adaptive control schemes for mobile robot formations with triangularised structures," *IET Control Theory Appl.*, vol. 4, no. 9, pp. 1817–1827, Sep. 2010.

[25] S. Zhao and D. Zelazo, "Bearing rigidity and almost global bearing-only formation stabilization," *IEEE Trans. Autom. Control*, vol. 61, no. 5, pp. 1255–1268, May 2016.

[26] M. Aranda, G. López-Nicolás, C. Sagüés, and M. M. Zavlanos, "Distributed formation stabilization using relative position measurements in local coordinates," *IEEE Trans. Autom. Control*, vol. 61, no. 12, pp. 3925–3935, Dec. 2016.

[27] E. A. Macdonald, "Multi-robot assignment and formation control," M.S. thesis, Dept. Elect. Comput. Eng., Georgia Inst. Technol., Atlanta, GA, USA, 2011.

[28] R. Falconi, S. Gowal, and A. Martinoli, "Graph based distributed control of non-holonomic vehicles endowed with local positioning information engaged in escorting missions," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 3207–3214.

[29] A. Widyotriatmo, E. Joelianto, A. Prasdianto, H. Bahtiar, and Y. Y. Nazaruddin, "Implementation of leader-follower formation control of a team of nonholonomic mobile robots," *Int. J. Comput. Commun. Control*, vol. 12, no. 6, pp. 871–885, 2017.

[30] Z. Lin, L. Wang, Z. Han, and M. Fu, "Distributed formation control of multi-agent systems using complex Laplacian," *IEEE Trans. Autom. Control*, vol. 59, no. 7, pp. 1765–1777, Jul. 2014.

[31] S. Gowal and A. Martinoli, "Real-time optimization of trajectories that guarantee the rendezvous of mobile robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 3518–3525.

[32] J. Yu and M. LaValle, "Distance optimal formation control on graphs with a tight convergence time guarantee," in *Proc. IEEE 51st Conf. Decis. Control*, 2012, pp. 4023–4028.

[33] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with optimality bounds," in *Redundancy in Robot Manipulators and Multi-Robot Systems*. Berlin, Germany: Springer, 2013, pp. 167–181.

[34] D. Miklic, S. Bogdan, S. Nestic, and R. Fierro, "A discrete grid abstraction for formation control in the presence of obstacles," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2009, pp. 3750–3755.

[35] J. Yu, "Constant factor time optimal multi-robot routing on high-dimensional grids," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, Jun. 2018, doi: 10.15607/RSS.2018.XIV.013.

[36] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 1917–1922.

[37] S. Tang and V. Kumar, "A complete algorithm for generating safe trajectories for multi-robot teams," in *Robotics Research*. Berlin, Germany: Springer, 2018, pp. 599–616.

[38] S. Tang and V. Kumar, "Mixed integer quadratic program trajectory generation for a quadrotor with a cable-suspended payload," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 2216–2222.

[39] Y. Kantaros and M. M. Zavlanos, "Distributed intermittent connectivity control of mobile robot networks," *IEEE Trans. Autom. Control*, vol. 62, no. 7, pp. 3109–3121, Jul. 2017.

[40] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2008, pp. 1928–1935.

[41] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*. Berlin, Germany: Springer, 2011, pp. 3–19.

[42] J. Alonso-Mora, P. Beardsley, and R. Siegwart, "Cooperative collision avoidance for nonholonomic robots," *IEEE Trans. Robot.*, vol. 34, no. 2, pp. 404–420, Apr. 2018.

[43] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, on-line collision avoidance for dynamic vehicles using buffered Voronoi cells," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 1047–1054, Apr. 2017.

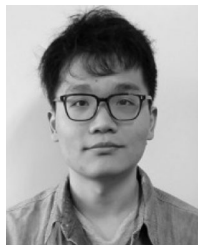
[44] V. R. Desaraju and J. P. How, "Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 4956–4961.

[45] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Trans. Robot.*, vol. 33, no. 3, pp. 661–674, Jun. 2017.

[46] L. Chaimowicz, N. Michael, and V. Kumar, "Controlling swarms of robots using interpolated implicit functions," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2005, pp. 2487–2492.

[47] P. Molnár and J. Starke, "Control of distributed autonomous robotic systems using principles of pattern formation in nature and pedestrian behavior," *IEEE Trans. Syst., Man, Cybern., B (Cybern.)*, vol. 31, no. 3, pp. 433–435, Jun. 2001.

[48] M. M. Zavlanos and G. J. Pappas, "Potential fields for maintaining connectivity of mobile networks," *IEEE Trans. Robot.*, vol. 23, no. 4, pp. 812–816, Aug. 2007.
[49] S. LaValle and J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics*. Wellesley, MA: AK Peters/CRC Press, 2001, pp. 303–307.
[50] K.-H. Tan and M. A. Lewis, "Virtual structures for high-precision cooperative mobile robotic control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 1996, vol. 1, pp. 132–139.
[51] M. Gauci, M. E. Ortiz, M. Rubenstein, and R. Nagpal, "Error cascades in collective behavior: A case study of the gradient algorithm on 1000 physical agents," in *Proc. 16th Conf. Auton. Agents MultiAgent Syst.*, 2017, pp. 1404–1412.
[52] M. Gauci, R. Nagpal, and M. Rubenstein, "Programmable self-disassembly for shape formation in large-scale robot collectives," in *Distributed Autonomous Robotic Systems*. Berlin, Germany: Springer, 2018, pp. 573–586.
[53] J. McLurkin, J. Smith, J. Frankel, D. Sotkowitz, D. Blau, and B. Schmidt, "Speaking Swarmish: Human-robot interface design for large swarms of autonomous mobile robots," in *Proc. AAAI Spring Symp.: To Boldly Go Where No Human-Robot Team Has Gone Before*, 2006, vol. 72, pp. 72–75.
[54] M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 3293–3298.



Hanlin Wang received the B.E. degree in mechanical engineering from Shanghai Jiao Tong University, Shanghai, China, in 2015. He is currently working toward the Ph.D. degree in computer science with the Department of Computer Science, Northwestern University, Evanston, IL, USA.

His research interests include swarm systems, planning and scheduling, and distributed computation systems.

Michael Rubenstein received the Ph.D. degree in computer science from the University of Southern California, Los Angeles, CA, USA, in 2009.

He is currently an Assistant Professor working on swarm robotics and control with the Department of Computer Science as well as the Department of Mechanical Engineering, Northwestern University, Evanston, IL, USA. His research interests include robot swarms and multirobot systems.