A Provably-Correct Decentralized Lattice Formation Algorithm for Multi-Robot Systems

Yang Song and Jason M. O'Kane

***Abstract*— We describe a new decentralized algorithm for multi-robot systems to form arbitrary repeated lattice patterns. In our prior work, we showed how to represent the desired lattice pattern using a directed graph in which each edge is labeled with a rigid body transformation, and proposed an algorithm that accepts this graph as input and computes destinations for each robot using only local information. In this paper, we designed a novel algorithm to resolve several limitations of the existing algorithm using a new message passing procedure and a new movement strategy.**

We prove that, by executing this algorithm, the robots will form the desired lattice pattern in a bounded amount of time. We further show that, if the robots' communication graph is connected at the start of the algorithm, it will remain connected throughout the algorithm's execution. Using a simulation, we demonstrate that this algorithm works correctly for systems with dozens of autonomous robots to form various lattice patterns. Moreover, the experiments show a significant improvement in formation quality for our new algorithm compared to our previous approach.

I. Introduction

Multi-robot systems have attracted a growing number of research efforts in recent years. In particular, the problem of creating formations of multiple robots been considered in multiple contexts, including autonomous ground, aerial, and underwater vehicles. Existing methods use either centralized or distributed algorithms to solve certain formation problems, subject to different constraints such as robot models, communication limits, *etc.*. In our previous work, we proposed a decentralized algorithm to solve a lattice pattern formation problem for the multi-robot systems, in which each robot makes decisions based only local information [10].

The input to our previous algorithm is a representation of desired formation in the form of a directed graph in which each vertex represents a "role" for a robot to play in the formation, and each outgoing edge is labeled with rigid body transformation indicating the desired location of one of that robot's neighbors. This kind of graph is able to represent both finite and infinitely repeating formations. As the algorithm proceeds, each robot continually broadcasts messages to the other nearby robots. Based on these messages, the robots organize themselves into a rooted tree structure, and then move to the positions assigned to them by their parents in this tree.

Although this algorithm performed reasonably well in our simulated experiments, it has two critical limitations. First, there is no guarantee that the robots will eventually form

Yang Song (song24@email.sc.edu) and Jason M. O'Kane (jokane@cse.sc.edu) are with the Department of Computer Science and Engineering, University of South Carolina, Columbia.

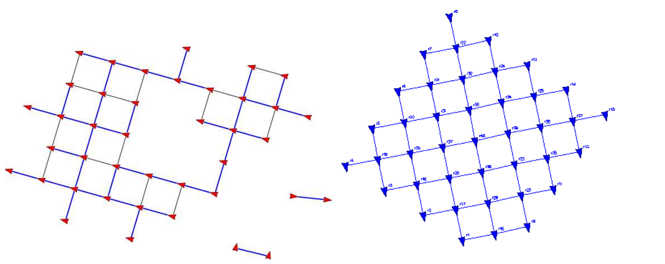


Fig. 1: Simulation results for 40 robots started from the same random but connected distribution of poses. The robots formed a repeating square pattern using both our our prior algorithm (left) and the new algorithm presented in this paper (right). The figure shows the final positions resulting from each algorithm.

the desired formation and terminate the algorithm. Instead, the robots may oscillate under some conditions. Second, the motion strategy employed for "orphan" robots—those to whom the chosen parent cannot assign a role—was simply to move directly away from that parent. This motion strategy may cause the formation become disconnected, decreasing the overall solution quality. In this paper, we introduce a new algorithm that resolves these limitations. Figure 1 shows an example of a repeating square pattern formed by 40 robots using our previous algorithm and new algorithm, respectively.

Specifically, this paper makes three new contributions beyond our prior work.

1) We describe a new algorithm which, while sharing some similarity to our existing work, differs in two essential ways. First, we define a new procedure to construct a global spanning tree from the communication graph, focusing on the combinatorial relationship of robots rather than their geometric positions. This change prevents oscillations. Second, our new plan directs the only selected robot move to a place where its pose, with its neighbor's pose, contributes to the desired formation. Third, we introduce a new movement strategy to maintain persistent connectivity of the robots' communication. This change improves the quality of the final solution.
2) We prove that robots executing this new algorithm will form the desired lattice with bounded execution time. We also show that, if the communication graph is initially connected, then it will remain connected throughout the execution of the algorithm.
3) We present a series of simulations. These simulations demonstrate the effectiveness of our new algorithm in comparison to the prior algorithm for the same problem.

In the following sections, first we summarize related work on decentralized formation algorithms and control approaches in Section II. Then in Section III we describe the formation problem we are solving. Section IV gives a quick summary of the existing algorithm. We explain our new algorithm in detail, and prove the time-bounded performance of our algorithm in Section V. We demonstrate simulated experiments to verify our algorithm and compare the performance of both existing and new algorithms in Section VI. In the final section, we conclude the paper and discuss the outlook for future improvements to the algorithm.

II. Related Work

Many works on the multi-robot lattice formation problem used distributed methods. Fujibayashi, Murata, Sugawara and Yamamura proposed a probabilistic-based control algorithm to generate the triangular lattice pattern [2]. Hanada, Lee and Chong constructed an algorithm to form separated triangular lattices and then reunify them as a whole in an environment with obstacles [4]. Notably, approaches using artificial virtual forces, such as the physicomimetics framework (PF), were well-studied for repeated lattice patterns formation problem, such as triangular, square, and hexagonal lattices [2], [4]–[9], [11], [12]. Robots can form specific lattice patterns using attractive or repulsive forces to adjust their positions, based on only measurements of range and bearing to nearby robots. W. Spears, D. Spears, Heil, Zarzhitsky, and Hamann discussed the swarm intelligence and applied PF-based methods to the multiple particle robots systems to form hexagonal lattices [12]. Martinson and Payton extended the PF-based algorithm by adding an extra alignment step before applying the virtual force among robots. The robots first move to parallel lines using compasses and then form square lattice using PF without suffering from local minima [6]. Navarro, Pugh, Martinoli, and Matía implemented a PF-based algorithm as a finite state machine, thus, robots form the triangular lattice without assumption of holonomocity [8]. Based on the PF-based approach used for the triangular formation, Mullen, Monekosso, Barman, and Remagnino developed a cooperative formation control strategy for the hexagonal lattices pattern. Their robots coordinate to determine the centroids of the hexagonal cells and place virtual robot nodes in those positions to avoid local minima [7]. Prabhu, Li, and McLurkin also used artificial forces for the hexagonal lattice formation, they contributed to a local error correction algorithm to detect and correct the misplacement [9]. Collet and Fanchon refined the virtual force method by changing the interaction rules between different types of agents, so that the robots could group themselves into hexagonal tiles [1].

Nevertheless, a common limitation of above-mentioned work is that only one specific lattice pattern known to the algorithm designer can be formed by using each specific algorithm. In contrast, our algorithm accepts a graph representation of desired lattice pattern as part of its input.

We have observed that the disconnection of the robots' communication graph has a major negative impact on the final formation quality in our prior work [10]. To enforce the connectivity of robots' communication, an important idea we have in this paper is to maintain a global spanning tree constructed from local communication.

Habibi and McLurkin presented an approach to build a maximum-leaf spanning tree to guarantee the connectivity of robots' network communication [3]. The spanning tree, rooted at the goal location where all the robots should be recovered, provides each robot a path to the goal. Moreover, only the leaves of the tree are allowed to move during the recovery, whereas the internal nodes remain motionless as a navigation guide to lead the robot back to the root. We use a different strategy to construct a spanning tree of the communication graph. In contrast to their approach that drives maximum number of leaves to the goal, we contribute a motion plan that, in our spanning tree, only relocates one robot to a specified goal position, while generating small but necessary movements of the others to preserve connectivity.

III. Problem Statement

The formation algorithms we focus on in this paper share the same problem statement.

A. Robot Model

Consider a set of n homogeneous **robots** $R = \{r_1, \dots, r_n\}$ in an obstacle-free plane. We assign each robot $r \in R$ a unique, comparable **identifier (ID)**, denoted $\mathrm{id}(r)$. That is, any pair of robots can determine the numerical order of their IDs. For example, we can assign a robot an ID using its serial number or network MAC address.

1) Poses and coordinate frames

We represent the **pose** of each robot r_i using $p_i = (x_i, y_i, \theta_i)$, which consists of the robot's position and orientation, expressed in an arbitrary but fixed global coordinate frame.

From the view of a robot itself, a **body frame** is defined, in which the robot is always at the origin, facing along the positive x-axis. Note that each robot obtains observations and makes moving decisions in its own body frame, without knowing its own global coordinates in the environment. For explanation purpose, we use the global coordinates of robots to analyze our methods, a homogeneous matrix $\mathbf{T}(r_i)$ represents the rigid body transformation from the body frame of r_i to the global frame:

$$\mathbf{T}(r_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & x_i \\ \sin\theta_i & \cos\theta_i & y_i \\ 0 & 0 & 1 \end{bmatrix}.$$

2) Motion, sensing, observation, and communication

We assume that each robot moves with a bounded linear velocity $v \in [0, v_{\max}]$ and can rotate in place with unbounded angular velocity.

We use a disk to represent the scope in which a robot can sense and communicate with other robots. Call the radius of this disk the robots' **range** and denote it as ϕ. The other robots within this range are called its **neighbors**. Assume

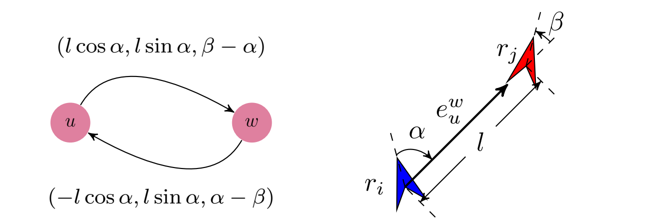

$(l\cos\alpha, l\sin\alpha, \beta-\alpha)$
u
w
$(-l\cos\alpha, l\sin\alpha, \alpha-\beta)$
β
r_j
e_u^w
α
l
r_i

Fig. 2: [left] Two nodes u, w in a lattice graph. The edge from u to w reflects the rigid body transformation between the two: a robot in role w is at pose $(l\cos\alpha, l\sin\alpha, \beta-\alpha)$ relative a robot in role u. [right] Poses of robots r_i and r_j that satisfy this lattice graph, with role functions $f(r_i) = v, f(r_j) = w$.

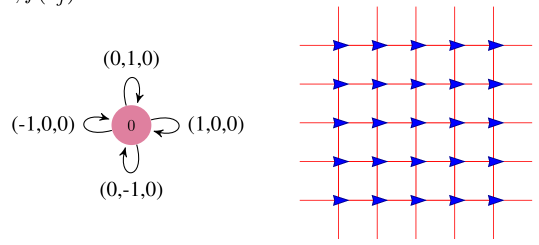


Fig. 3: [left] A lattice graph representing a repeated square pattern. There is one node with four outgoing edges connecting to itself. Here the edge length is 1. For a robot playing the role of this node, its neighbors should be in locations $(1,0), (0,-1), (-1,0)$ and $(0,1)$, with the same orientation. [right] Repeated square lattice pattern formed in terms of the left graph.

that the sensing and communication occurs in discrete time steps, so that every Δt seconds:

1) Each robot collects **observations** indicating the IDs and relative poses of its neighbors.
2) Each robot broadcasts a **message** to its neighbors. Details about the specific messages used in our algorithm appear in Section V.

In practice, one can choose for ϕ either the robots' effective communication range or the robots' effective sensing range, whichever is smaller.

Two assumptions here, namely synchronous clocks and unbounded angular velocity, would of course be very difficult to realize in a practical multi-robot system. These assumptions are used to simplify the the connectivity analysis in Section V, and are not crucial to the correctness of the algorithm.

B. Lattice graph

We solve the multi-robot lattice formation problem using a graph representation of desired lattice pattern that the robots should form [10].

Definition 1: A ***lattice graph*** *is a strongly connected directed multi-graph in which each edge e is labeled with a rigid body transformation $\mathbf{T}(e)$ and each $v \xrightarrow{\mathbf{T}(e)} w$ has an inverse edge $w \xrightarrow{\mathbf{T}(e)^{-1}} v$ [10].*

The intuition is that, when the lattice is constructed, each robot will be associated with one lattice graph vertex, and that the outgoing edges of that vertex will correspond to the number and relative poses of that robot's neighbors. For example, a repeated square lattice pattern can be represented by a lattice graph in Figure 3.

Definition 2: Given a lattice graph $G = (V, E)$ and a set of robots $R = \{r_1, \ldots, r_n\}$, we say that R ***satisfies*** *G if there exists a function $f : R \to V$ that preserves the neighborhood*

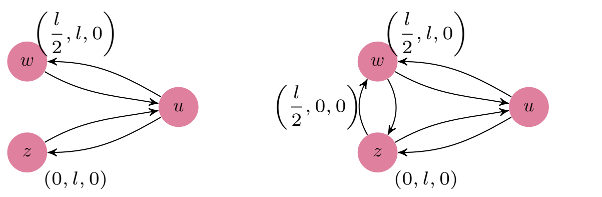

$\left(\frac{l}{2}, l, 0\right)$
w
u
z
$(0, l, 0)$
$\left(\frac{l}{2}, l, 0\right)$
$\left(\frac{l}{2}, 0, 0\right)$
w
u
z
$(0, l, 0)$

Fig. 4: [left] A lattice graph that is not self-consistent when $\phi > l$. The distance between robots with roles w and z is less than $l/2 < \phi$, but no edge connects w and z. [right] A lattice graph that is self-consistent when $\phi > l$.

structure of G. Specifically, for any i and j, if r_i and r_j are neighbors, there must exist an edge $e_u^w : f(r_i) \longrightarrow f(r_j)$ in E, such that $f(r_i) = u$, $f(r_j) = w$, and $\mathbf{T}(r_j) = \mathbf{T}(r_i)\mathbf{T}(e_u^w)$. That is, we require the transformation $\mathbf{T}(e_u^w)$ associated with edge e_u^w describe the relative pose of r_j in the body frame of r_i. See Figure 2.

However, the graph definition of a lattice may describe some geometric patterns contradicting themselves. To represent a pattern properly using a lattice graph, we define a constraint, termed "self-consistent", for a lattice graph, so that the represented pattern will not contradict itself by forcing pairs of robots to be mutual neighbors, with roles that are not adjacent in the lattice graph. Figure 4 illustrates this idea, showing a comparison between a lattice graph that is not self-consistent with one that is self-consistent.

Definition 3: Given a range $\phi > 0$, a lattice graph is ***self-consistent*** *for this range if, for any two paths with the same starting node,*

$$u \xrightarrow{\mathbf{T}(e_u^k)} \cdots \xrightarrow{\mathbf{T}(e_m^w)} w, \text{ and } u \xrightarrow{\mathbf{T}(e_u^j)} \cdots \xrightarrow{\mathbf{T}(e_n^z)} z,$$

for which the distance between two ending nodes is less than or equal to ϕ, there exist edges between w and z.

This definition is important because our algorithm is correct only when the input lattice graph is self-consistent. We discuss the surprisingly challenging problem of of determining whether a given lattice graph in Section VII.

C. Evaluation criteria

We evaluate our algorithms by measuring both the execution time and the final formation quality.

1) Execution time

Ideally, we want a robot to move to a position and stay there to be a part of desired formation, after executing the algorithm for a while.

Definition 4: A robot r_i is ***static*** *at time t if its pose p_i remains the same at all future times, so that*

$$p_i(t') = p_i(t) \quad \text{for all } t' > t.$$

Definition 5: The ***execution time*** *T of the system is the smallest time at which all of the robots are static:*

$$T = \inf\{t \in (0, \infty) \mid \text{every } r \in R \text{ is static at time } t\}$$

The intuition is, the smaller the T is, the more efficient the algorithm is.

r_1
r_2
$\text{dist}(r_1, r_2) = 10 \leq \phi$
r_3
r_4
$\text{dist}(r_3, r_4) = 15 > \phi$

Fig. 5: Consider a special instance of lattice graph in Figure 2, assume the range $\phi = 10$, edge length $l = 10$, $\alpha = 3\pi/4, \beta = \pi/4$. [left] Two robots r_1, r_2 satisfy the lattice graph with $Q = 1$. [right] Two robots r_3, r_4 can not observe each other, but they satisfy the lattice graph with $Q = 0$.

2) *Fulfillment Ratio*

We use a simple criteria to evaluate the quality of the formed lattice pattern, called **fulfillment ratio**.

For a robot r_i, let N_i denote the number of neighbors of r_i when it is static, and let E_i denote the number of outgoing edges of its role vertex in the lattice graph, that is, the maximum number of neighbors it could have in order to form the desired lattice represented by given lattice graph.

The overall formation quality is measured by finding the average fulfillment ratio:

$$Q = \frac{1}{n} \sum_{i=1}^{n} \frac{N_i}{E_i}.$$

The ratio $Q \in [0, 1]$ approximately reflects how satisfactory the final formation is with regard to the lattice graph. We prefer a formation with a value of Q as large as possible. Figure 5 shows different cases when two robots satisfy the lattice graph in Figure 2. Also, in Figure 1, the fulfillment ratio of the left formation $Q = 0.6375$, whereas the right formation, with its fulfillment ratio $Q = 0.7875$, has a better quality than the left one.

IV. Review of Existing Algorithm

In our previous work [10], we introduced a decentralized solution to solve the problem introduced in Section III via local task assignment. There are three major properties of the prior algorithm.

1) It is a stateless algorithm. Robots repeatedly organize themselves by constructing a collection of tree structures using the information from the incoming messages on-the-fly.
2) Multiple robots move toward their destinations simultaneously.
3) There is no guarantee that the robots will reach positions satisfying the lattice graph within finite time.

A robot remains motionless if it acts as a root in the tree structure, otherwise, it moves toward to the destination assigned by its parent in the tree, while maintaining the communication with its parent. Our new algorithm borrows one essential lemma from the prior approach.

Lemma 6 (Bounded range [10]): If robot r_i is the neighbor of robot r_p at time t, and both robots move with maximum velocity v, then r_i will still be a neighbor of r_p at time $t + \Delta t$ if

$$||p_i(t + \Delta t) - p_p(t)|| \leq \phi - v\Delta t.$$

The idea is that, to ensure that a robot r_i remains within the range of its parent r_p at next stage, it must move to a position with distance at most $\phi - v\Delta t$ of the current position of r_p.

*Definition 7: Given a robot r_i at pose p_i, with linear velocity v, the **bounded range** of r_i at time $t+\Delta t$, denoted as $\mathcal{B}_i$, is a circular region centered at p_i, with radius $\phi - v\Delta t$.*

The intuition is that, any robot that intends to remain within range of r_i at the next time step must move into (or stay within) the bounded range of r_i.

V. Algorithm Description

This section explains our new work on a stateful algorithm. The basic idea is to construct the desired lattice one element at a time. The robot with the highest ID, forms a "root" for the lattice. From there, the robot with the second highest ID moves to be a neighbor of the first robot, assuming one of the neighbor positions defined by the lattice graph. Meanwhile, other robots only move, if necessary, to maintain the connectivity of communication graph. By repeating this process, we sequentially drive one robot at a time to vacancies in the emerging formation, until all the robots are in place. The primary challenges arise from the fact that each robot must execute this strategy using only local information, and from the need to maintain connectivity across all robots through the entire process.

A. *Stable versus Unstable*

Each robot maintains a boolean state variable, tracking whether it is **stable** or **unstable**. Each robot begins in the unstable state, and changes permanently to the stable state when either the robot determines that it has the highest ID over all robots (see Section V-B), or the robot finishes navigating to an open vacancy (see Section V-C). The intuition is that stable robots remain motionless, whereas unstable robots are either moving toward a vacancy or waiting for their turn do so.

The core of our algorithm is for the unstable robot with the highest ID to position itself as a neighbor of the stable robot with the highest ID, that does not yet have the maximum number of neighbors allowed by its role in the lattice graph.

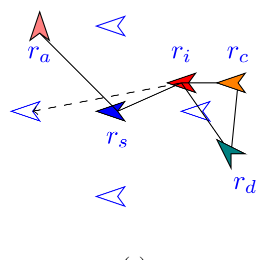
*Definition 8: Let r_s denote a stable robot and let u denote a role vertex u for r_s. For each out-edge e_u^w of u in the lattice graph, there is a pose $\hat{p}$ corresponding to the vertex w. If there is no stable robot at $\hat{p}$, then $(\hat{p}, w)$ is a **vacancy** of r_s. (If, by chance, there is an* unstable *robot at $\hat{p}$, we still consider $(\hat{p}, w)$ a vacancy of r_s.)*

Figure 6(a) shows an example, in which the robot r_s is the only stable robot and has four vacancies computed in terms of the lattice graph in Figure 3. The robot with the second highest ID, r_i, is moving toward the vacancy directly ahead of r_s. Robot r_i will change its state from unstable to stable after it reaches this pose.

B. *Communication*

We assume that the robots communicate by broadcasting messages to their neighbors every Δt seconds. The main objective for this communication is to retrieve some principal global information of the system and then to construct a spanning tree of the communication graph.

The principal global information includes: the highest ID among all stable robots who have at least one vacancy, as



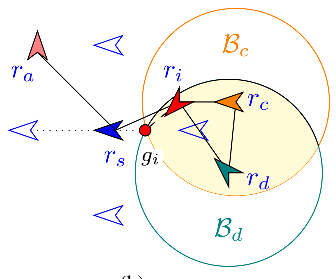
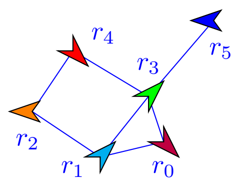


Fig. 6: (a) Robot r_s be a stable robot, it has two neighbors r_a, r_i and four vacancies. Robot r_i is the relocate robot and its ultimate destination is the vacancy (hollow) ahead of r_5. (b) The relocate robot r_i has two descendants r_c, r_d in the moving subtree. Two circles denote the bounded ranges of r_c, r_d, r_i finds an intermediate goal (red dot) $g_i \in (\mathcal{B}_c \cap \mathcal{B}_d)$.



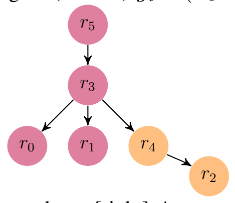


Fig. 7: [left] Communication graph of five robots. [right] Assume r_5 has the highest stable ID and a vacancy. A spanning tree corresponding to the left communication graph is built, in which the root is r_5. Let r_4 be the relocate robot, it then is the root of the moving subtree, in which r_2 is its descendant.

well as one of corresponding vacancies, called the **most important vacancy**, and the highest ID among all unstable robots. The stable robot who has the highest ID and a vacancy will be elected as the root of the spanning tree, call it the **root robot**. The unstable robot who has the highest ID, termed the **relocate robot**, should relocate to the most important vacancy.

We define a message type containing data, as Table I lists, for robots to acquire the principal global information. The purpose of tracing the paths from the root and the relocate robots is to prevent the "rumor," in which a robot relies upon the information generated originally by itself. Specifically, a robot rejects information for which its own ID appears in the "pathFromRoot" or "pathFromRelocate" lists.

Data	**Description**
senderID	ID of the robot that sent this message
highestID	highest robot ID in the system
rootID	ID of the root robot
relocateID	ID of the relocate robot
parentID	ID of the sender robot's spanning tree parent
vacancy	the most important vacancy of the root robot
subtree	a boolean marker showing if the sender robot is in the subtree rooted at the relocate robot
stable	a boolean label which indicates if the sender robot is stable
pathFromRoot	a list of robot IDs who are in the shortest path to the root robot
pathFromRelocate	a list of robot IDs who are in the shortest path to the relocate robot

TABLE I: Data included in each message.

We establish a "communication timeout" parameter that measures the time needed for all the robots to reach consensus about the principal global information.

Definition 9: The ***communication timeout*** *is the number of time steps that the robot needs to wait before it is confident about the principal information from the incoming messages.*

Choosing a suitable value for the communication timeout depends on both the initial distribution of the robots' poses and the shape and dimensions of the final formation. In the worst case, the communication timeout can be bounded above by a time linear in the number of robots.

Our algorithm uses the communication timeout in two related ways. First, each robot waits until the communication timeout has passed since the last change to the highest ID that the robot knows about. When this occurs, the robot can correctly decide whether it is or is not the highest ID robot. The robot that does have the highest ID can then become stable. Second, the robots use the communication timeout in a similar way to identify the unstable robot with the highest ID. This robot then becomes the relocate robot, and begins navigating toward the most important vacancy.

Algorithm 1 briefly illustrates how each robot computes its message based on the incoming messages from this neighbors. Each robot can determine the highest ID by repeatedly comparing its own ID with the highest IDs transmitted by its neighbors. The robots compute the highest ID unstable robot and the highest ID stable robot with vacancy in a similar way.

During the same communication process, the robots also construct a spanning tree of their communication graph, rooted at the highest ID stable robot with vacancy. Each robot that has at least one stable neighbor, chooses for its parent in this tree the stable neighbor that is the fewest hops from the root. Each robot with no stable neighbors also chooses a parent to minimize hops to the root, but in this case, that neighbor will necessarily be unstable. A special exception, designed to reduce unnecessary movements, is that no robot chooses the relocate robot as its parent, if it has another neighbor with the same or fewer hops to the root. Ties, if any, are broken in favor of the highest ID. Figure 7 shows an example.

Because our movement strategy uses the parent-child relationships in this spanning tree to maintain connectivity, each robot also keeps track of whether the relocate robot is its ancestor in the spanning tree, denoted "subtree" in the message type.

C. Movement Strategy

We use the bounded movement idea from Lemma 6 to maintain the connection between two robots if there exists a parent-child relationship between them in the tree structure. Only the relocate robot and its descendants (if any) move. A key difference from the prior algorithm is an inversion of responsibility: rather than tasking each robot in this subtree to remain within range of its parent, we instead require each parent to obey a "No Child Left Behind (NCLB)" policy, which allows only movements that are guaranteed to stay within bounded range of each of its children. In turn, each child moves close enough to its parent to enable the parent to make progress in spite of the NCLB constraint.

Specifically, each moving robot computes in the intersection of of the bounded ranges of its children. This intersection region forms a "safe region" such that if the robot stays with its safe region, it can guarantee to remain within range of

```
input : inMessages, stable
output: myMsg
myMsg ← empty message;
myMsg.senderID ← myID;
myMsg.highestID ← myID;
myMsg.stable ← stable;
myVacancy ← getVacancy();
myMsg.rootID ← (stable and myVacancy ≠ null) ? myID : INT_MIN;
myMsg.relocateID ← (not stable) ? myID : INT_MIN;
eligibleMsgs ← {};
vacancyFromMsgs ← null;
foreach inMsg in inMessages do
    myMsg.highestID ← max(myMsg.highestID, inMsg.highestID);
    if not inMsg.pathFromRoot.contains(myID) and not
    inMsg.pathFromRelocate.contains(myID) then
        insert inMsg into eligibleMsgs;
        if inMsg.rootID > myMsg.rootID then
            myMsg.rootID ← inMsg.rootID;
            vacancyFromMsgs ← inMsg.vacancy;
        end
        myMsg.relocateID ← max(myMsg.relocateID, inMsg.relocateID);
    end
end
minRootPath ← getShortestPathFromRoot(eligibleMsgs);
minRelocatePath ← getShortestPathFromRelocate(eligibleMsgs);
myMsg.vacancy ← (myID = myMsg.rootID) ? myVacancy :
vacancyFromMsgs;
myMsg.pathFromRoot ← minRootPath.append(myID);
myMsg.pathFromRelocate ← minRelocatePath.append(myID);
if myID ≠ myMsg.rootID then
    parentMsg ← getParentMsg(eligibleMsgs, myMsg.pathFromRoot);
    myMsg.parentID ← parentMsg.senderID;
    myMsg.subtree ← myID = myMsg.relocateID or parentMsg.subtree;
end
return myMsg;
```

Algorithm 1: Compute message.

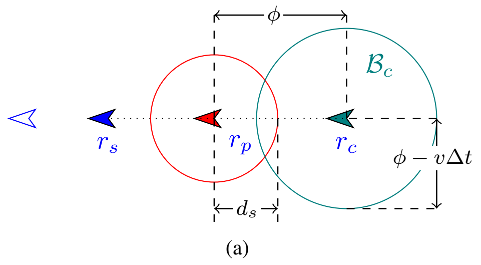
all of its children in the next time step. A robot that stays within its safe region is said to obey NCLB.

The movement strategy is divided into three cases.

1) If a robot is stable, or it is not in the moving subtree, then it does not move.
2) If a robot is the relocate robot, it moves, without violating NCLB, toward its parent, unless its parent is the root, in which case it moves toward the vacancy. Figure 6(b) shows a scenario in which a relocate robot computes an intermediate goal according to the bounded ranges of its children.
3) For other robots in the moving subtree, we set a "safe distance", denoted $d_s \in (0, \phi - v\Delta t)$. If the robot is within distance d_s of its parent, it does not move. Otherwise, moves it directly toward its parent while staying within its NCLB safe region. Figure 8 gives another example of the NCLB movement strategy, in which the relocate robot first waits for its child to come closer, then computes its intermediate destination in both bounded ranges of its children.

A special case occurs if the safe region for any robot is empty. In that case, the robot does not move. We can still guarantee connectivity in this case because the children (which must exist to get an empty safe region) will only move toward the robot. Movements that decrease the distance between robots cannot break their connectivity. If $d_s < \phi - v\Delta t$, then the safe region will eventually be non-empty.

In this way, the relocate robot navigates to the vacancy. It then changes its state from unstable to stable, triggering



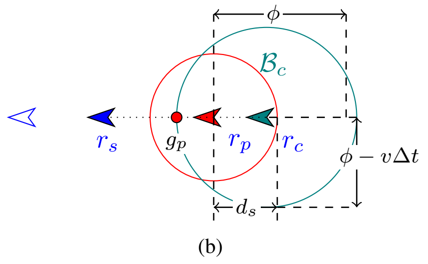


Fig. 8: (a) The relocate robot r_p intends to move to the vacancy (hollow arrow) of the root robot r_s. Robot r_c is a child of r_p. Since $\text{dist}(r_p, r_c) > d_s$, r_p waits for r_c to move close to it. (b) When $\text{dist}(r_p, r_c) \le d_s$, the relocate robot r_p finds the closest point $g_p \in \mathcal{B}_c$ (red dot) to the vacancy as an intermediate goal.

the communication process to identify the next relocate robot and the next vacancy. After each robot has moved to a vacancy, the formation is complete, and the algorithm terminates upon discovering that no unstable robots remain.

D. Correctness

Now we prove that robots form the desired lattice pattern correctly, in a finite number of steps using our algorithm.

Lemma 10: Given a set of robots $R = \{r_1, \ldots, r_n\}$, if the communication graph is connected at time t, then using our motion strategy, the communication graph stays connected at time $t + \Delta t$.

Proof: The parent-child relationships employed by the algorithm constitute a spanning tree of the communication graph. Consider each edge in that spanning tree, connecting a robot r_p to its child r_c. Section V-C has already argued that, under our motion strategy, r_p will not make any movements that enable r_c be more than ϕ distance from r_p away at the next time step. Therefore, the communication graph at time $t + \Delta t$ contains an edge connecting r_p to r_c. The collection of all such edges constitutes a spanning tree for the communication graph at time $t + \Delta t$. Because that graph has a spanning tree, it must be a connected graph. ☐

With lemma 10, we are certain that the robots' communication graph remains connected from an initial connected one. As a result, the relocate robot can reach a vacancy by following the its path to the root.

Now we show that it takes bounded time for a relocate robot to reach a vacancy, where the location and the root robot's position satisfy the input lattice graph.

Theorem 11: Given a set of robots $R = \{r_1, \ldots, r_n\}$, if the communication graph is initially connected, robots will reach positions satisfying the input lattice graph with bounded time steps.

Proof: Our algorithm sequentially drives a robot, in a decreasing order of their IDs, to a vacancy until no more unstable robots remain in the system. Since the robot with the highest ID never moves, there are maximum $n-1$ robots to be relocated. An upper bound of the total execution time of the system could be:

$$T = \sum_{i=1}^{n-1} (T_i) + O(n),$$

in which the trailing $O(n)$ arises from the communication timeout needed for the highest ID robot to become stable. The total time for r_i, T_i, is bounded by the sum of the communication timeout, and the time spent actually moving to the vacancy, plus the total time waiting for at most $(n-1-i)$ descendants in the moving subtree. A loose upper bound on T_i is:

$$\begin{aligned} T_i &= T_{\text{timeout}} + T_{\text{drive}} + T_{\text{wait}} \\ &\le O(n) + i\phi/v + \phi(n-i-1)/v \\ &\le O(n) + (n-1)\phi/v \\ &= O(n). \end{aligned}$$

This bound derives from three observations. First, recall that the time complexity of the communication time T_{timeout} is generally $O(n)$ (Section V-B). Second, note that the distance traveled by r_i from its current position to the vacancy is approximately equal to the geometric length of its path from the root. Because each robot appears at most once on that path, we have $T_{\text{drive}} \le i\phi/v$. Finally, the maximum waiting time achieves its worst case when all $n-i-1$ other unstable robots are descendants and they form a linear chain with successive robots distance ϕ apart. In this case, the descendant robots need to move close to their parents, starting from leaf node, one-by-one. Thus, the waiting time is bounded by $T_{wait} \le (n-i-1)\phi/v$.

Therefore, the robots to reach positions satisfying the input lattice graph in $O(n^2)$ steps. □

VI. Implementation and Results

We implemented our algorithms with ROS and conducted a series of simulations to test the performance of both the old and the new algorithms. We have tested three types of repeated lattice patterns: square (Figure 3), hexagon, and octagon-square in an obstacle-free environment.

For each pattern, we varied the number of robots n between 10 and 40 in increments of 10. For each set of n robots, we repeated 10 trials of the experiments with different distributions of the initial poses. Also, the initial poses are randomly generated but with communication graph connected. For the new algorithm, we set both of the communication timeout and the movement timeout as the same as the number of robots. We measured the execution time of both algorithms using the simulation steps as unit. Figures 9, 10, 11 show the experiments results for three lattice patterns.

From these results, we conclude that the execution time for both algorithms scales quadratically as the number of robots increases, confirming our analysis. Current experimental results indicate that the new algorithm takes more simulation steps for the robots to reach static states than the old algorithm. The first reason is that when executing the new algorithm, every robot needs a communication timeout to confirm the change of its states, and an movement timeout if it is supposed to move. Moreover, each timeout scales with the size of the robots set. The second reason is that when using the new algorithms, only one robot moves to a vacancy most of the time. Another reason is that the old algorithm is locally optimal with the sum of traveled distance, whereas the new algorithm only considers the combinatorial relationship of the robots.

On the other side, we are confident to get a better formation quality by executing the new algorithm, compared with the results by using the old algorithm. Since our new algorithm always outputs a deterministic combinatorial relationship of the given set of robots, regardless of their initial poses distributions. Therefore, these experimental statics show that the deviation of the fulfillment ratio is always zero when using the new algorithm.

Above all, there is a tradeoff between the execution time and the formation quality when comparing the performance of two algorithms. However, for the new algorithm, the execution time is proved bounded, but not for the old algorithm.

VII. Conclusion and Future Work

We presented a new decentralized multi-robot formation algorithm using only robots' local information inspired by our previous method. We proved that our algorithm guarantees to have robots reach positions satisfying the given lattice graph within bounded time. Moreover, our algorithm maintains the connectivity of the communication graph during its execution. Compared to the prior approach, the new method improves the formation quality notably.

We realized that it is difficult to determine if an arbitrary lattice graph is self-consistent by definition. For example, consider the lattice graph in Figure 3, if the rotation angle of an edge is $\sqrt{2}\pi/2$ instead of 0, it represents a whirlpool-like pattern, but is not self-consistent because there are no edges between some neighbor robots. Our future work consists of working on an algorithm to decide if a path could end at the same position as its starting position in finite steps.

Furthermore, the new algorithm has a good scalability. Current version only relocates one robot to one vacancy at a time. We anticipate to expand the algorithm so that multiple robots would be relocated to multiple vacancies at a time. Particularly, we expect to construct $k, 1 \le k < n$ spanning trees from the communication graph, by finding the k highest ID stable robots with vacancies, and k highest ID unstable robots in the system. As a result, along with a proper movement strategy, the execution time of the algorithm would be improved significantly.

References

[1] J. H. Collet and J. Fanchon, "Crystallization and tile separation in the multi-agent systems," *Physica A: Statistical Mechanics and its*

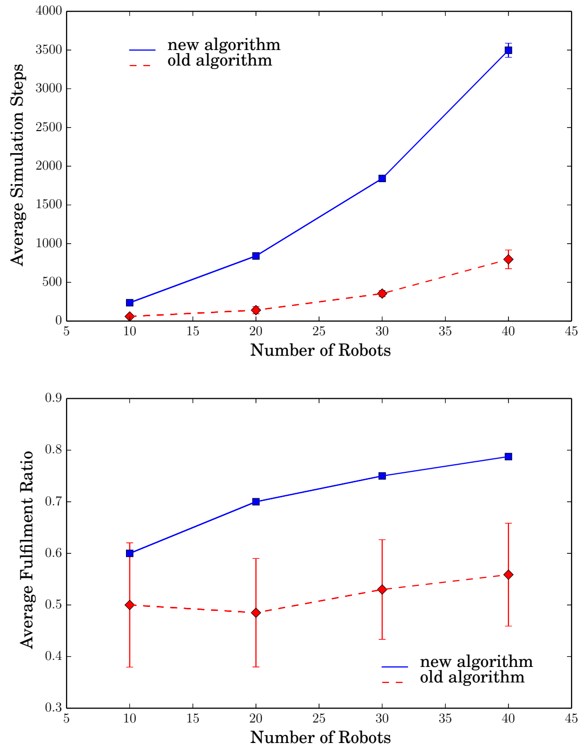


Fig. 9: Repeated squares pattern: [top] The average execution time with deviation. [bottom] The average fulfillment ratio with deviation.

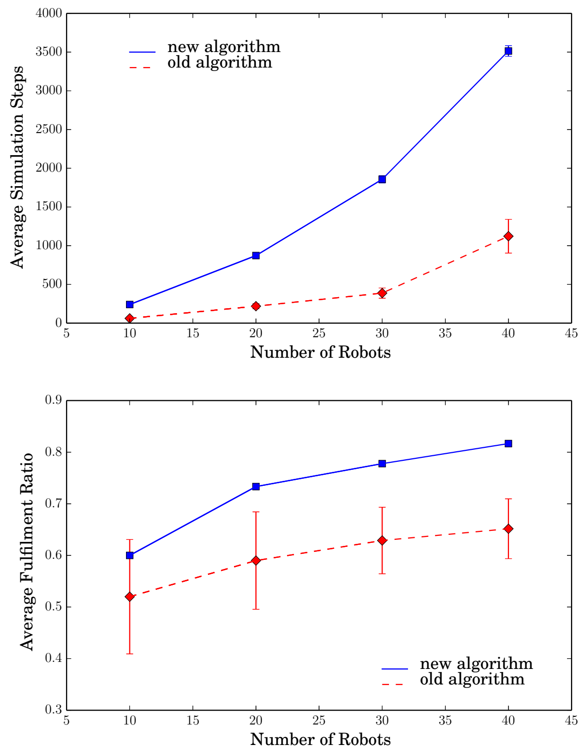


Fig. 10: Repeated hexagons pattern: [top] The average execution time with deviation. [bottom] The average fulfillment ratio with deviation.

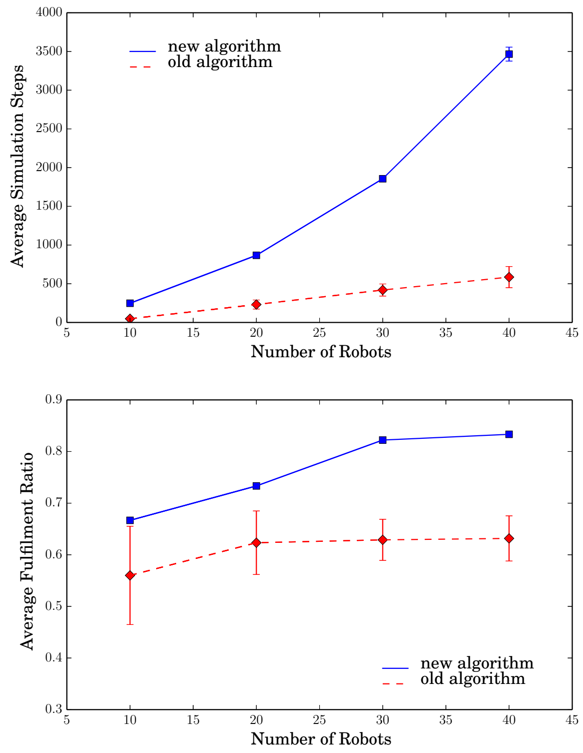


Fig. 11: Repeated octagons and squares pattern: [top] The average execution time with deviation. [bottom] The average fulfillment ratio with deviation.

Applications, vol. 436, pp. 405–417, 2015.

[2] K. Fujibayashi, S. Murata, K. Sugawara, and M. Yamamura, "Self-organizing formation algorithm for active elements," in *Reliable Distributed Systems, 2002. Proceedings. 21st IEEE Symposium on*. IEEE, 2002, pp. 416–421.

[3] G. Habibi and J. McLurkin, "Maximum-leaf spanning trees for efficient multi-robot recovery with connectivity guarantees," in *Distributed Autonomous Robotic Systems*. Springer, 2014, vol. 104, pp. 275–289.

[4] Y. Hanada, G. Lee, and N. Y. Chong, "Adaptive flocking of a swarm of robots based on local interactions," in *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, 2007, pp. 340–347.

[5] S. Lee, A. Becker, S. P. Fekete, A. Kröller, and J. McLurkin, "Exploration via structured triangulation by a multi-robot system with bearing-only low-resolution sensors," *Computing Research Repository*, vol. abs/1402.0400, 2014.

[6] E. Martinson and D. Payton, "Lattice formation in mobile autonomous sensor arrays," in *Swarm Robotics*. Springer, 2005, vol. 3342, pp. 98–111.

[7] R. J. Mullen, D. Monekosso, S. Barman, and P. Remagnino, "Reactive coordination and adaptive lattice formation in mobile robotic surveillance swarms," in *Distributed Autonomous Robotic Systems*. Springer, 2013, vol. 83, pp. 229–242.

[8] I. Navarro, J. Pugh, A. Martinoli, and F. Matía, "A distributed scalable approach to formation control in multi-robot systems," in *Distributed Autonomous Robotic Systems 8*. Springer, 2009, pp. 203–214.

[9] S. Prabhu, W. Li, and J. McLurkin, "Hexagonal lattice formation in multi-robot systems," in *Distributed Autonomous Robotic Systems*. Springer, 2014, vol. 104, pp. 307–320.

[10] Y. Song and J. M. O'Kane, "Decentralized formation of arbitrary multi-robot lattices," in *Proc. IEEE International Conference on Robotics and Automation*, Hong Kong, China, 2014.

[11] W. M. Spears, R. Heil, and D. Zarzhitsky, "Artficial physics for mobile robot formations," in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, 2005.

[12] W. M. Spears, D. F. Spears, J. C. Hamann, and R. Heil, "Distributed, physics-based control of swarms of vehicles." Springer, 2004, vol. 17, no. 2-3, pp. 137–162.