Decentralized Formation of Arbitrary Multi-Robot Lattices

Yang Song and Jason M. O'Kane

***Abstract*— In this paper, we propose a decentralized algorithm to form arbitrary repeating formations of multiple robots. Methods are known to form specific kinds of repeating structures such as squares, triangles, and hexagons by modeling each robot as a particle that responds to attractive and repulsive forces generated by nearby robots. However, such methods are generally designed by hand for one specific type of lattice. Our approach is more general, in the sense that we present a single algorithm, for which a description of the desired repeating pattern is part of the input. We represent this pattern as a directed graph, in which edges show the desired rigid body transformations between the local frames of pairs of neighbor robots. The robots autonomously organize themselves into a family of rooted trees, and use these trees to perform task assignments locally and without conflicts. We show, via our simulated implementation, that our algorithm works for robot systems with hundreds of robots to form various lattice patterns. Our experiments also show that the approach can recover rapidly from robot failures, even if those failures impact a large fraction of the robot population.**

I. Introduction

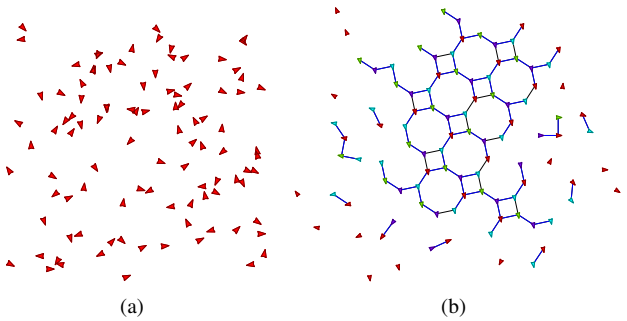
Large systems of autonomous robots can be useful for tasks ranging from surveillance and and search to the deployment of large-scale mobile sensor networks. Several scholars published broad reviews of this topic in early years [2], [3]. However, some robot coordination problems remain challenging, including the problems of pattern formation and adaptation [1].

Our work in this paper focuses on the multi-robot pattern formation problem. We propose a single decentralized algorithm that can form any repeating lattice pattern, with large numbers of robots. Figure 1 shows an example, in which 100 robots running our algorithm form a repeating octagon-square pattern.

The desired formation is represented using a graph called lattice graph that is known by every robot. Each vertex of the lattice graph represents one "role" that a robot can play. Because we are interested in repeating patterns, each role is likely to be filled by many different robots. The lattice graph edges, which are labeled with rigid body transformations, indicate the relative poses that neighboring robots should have in the completed lattice.

To run the algorithm, each robot only needs to sense the identities and relative poses of any nearby robots, and to broadcast short messages to those neighbors. Using this information, each robot continually executes a series of four steps.

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.



(a) (b)

Fig. 1. (a) The initial poses of 100 robots; (b) The final formation after executing our algorithm, using the octagon-square lattice graph shown in Figure 7.

1) First, each robot decides whether to consider itself a root robot—one that remains motionless—or a descendent robot—one that moves based on a task assignment from a parent robot. These decisions are made based on a concept we call "authority," which depends on the arbitrary IDs assigned to each robot, but also includes information about a robot's ancestors, to ensure that the robots quickly form a consistent, stable forest of authority trees. Each robot selects its highest-authority neighbor as its parent, or considers itself a root if its own intrinsic authority outweighs all of its neighbors.
2) Next, each robot chooses a role. Root robots always select, without loss of generality, the first lattice vertex as their role. Descendant robots accept their role as part of the task assignment from their parent.
3) After selecting a role, each robot computes a locally optimal task assignment for its neighbors, using the standard Hungarian algorithm [6]. Each robot broadcasts this assignment, along with the authority value used in the first step, to its neighbors.
4) Finally, each descendant robot moves toward to position assigned to it by its parent, subject to the constraint that it must remain within communication range with that parent along the way.

The result of this algorithm is that the robots can form different types of geometric formations, including the repeated lattice patterns.

One important feature of the algorithm is that it is *stateless*. Each robot chooses the movements to make and messages to send based strictly on its recent observations and messages received. The impact of this design is that the algorithm can recover rapidly from many kinds of unexpected changes, including changes to the set of active robots due to failures or additions.

The specific contributions of this paper are:

- We introduce a method for representing desired multi-robot formations—both repeating patterns and finite formations—using a directed multigraph called a *lattice graph*.
- We propose a strategy based on *authorities* for the robots to autonomously organize themselves into trees that show which robots will accept task assignments from which other robots, in a way that is robust to changes.
- Based on these trees, we use local task assignments to from the desired global formation based on local observations.
- We describe an implementation of this technique, and present experiments confirming its effectiveness.

The remaining presentation is organized as follows. We give a brief summary of previous works on decentralized coordination algorithms and distributed formation control strategies in Section II. The formation problem and necessary notations are addressed in Section III. In Section IV, we describe our algorithm in detail. Section V describes our implementation and the simulations to verify our algorithm and evaluate its performance. Concluding remarks appear in Section VI.

II. Related Work

Some scholars have worked on the repeated lattice pattern formation problem for multi-robot systems. Fujibayashi, Murata, Sugawara and Yamamura proposed decentralized a probabilistic-based control algorithm to generate the triangular lattice pattern [4]. Hanada, Lee and Chong constructed an algorithm to form separated triangular lattices and then reunify them as a whole in an environment with obstacles [5].

Specifically, triangle and hexagon lattice patterns can be formed using artificial virtual forces among robots [14] [15] [17] [18]. Spears, Heil and Zarzhitsky applied the physicomimetics framework (PF) to swarms of robots to form hexagonal formations [17]. The idea of PF is that the robots observe the relative positions of their neighbors and react to each other by attractive or repulsive forces in terms of these positions [18]. One limitation of using PF is that some robots may settle in local minima. Mullen, Monekosso, Barman and Remagnino created virtual robot node (VRN) architecture for the PF algorithm. Specific positions in the triangle lattice are replaced by virtual robot nodes so that finally a hexagon lattice could be formed [13]. The limitation of the VRN scheme is that robots need to coordinate to determine the center of the hexagonal cells.

Prabhu, Li and McLurkin extended Spears's work by coloring the robots and adding a local error correction strategy to avoid letting robot move to the center of a hexagonal cell due to the energy well. Hence, the formed repeated hexagonal lattice patterns are more stable [15]. However, purely using the PF-based algorithm limits the application on the holonomic robot model. Navarro, Pugh, Martinoli and Matía improved the PF-based strategy by implementing a finite state machine so that this algorithm can function with nonholonomic robots. As a result, the formed triangular lattice formation can move as a whole in the same direction [14].

In addition to triangular and hexagonal lattices, square lattice patterns can also be formed using PF. 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 [11].

Nevertheless, both probabilistic-based control formation algorithms and the methods based on force-balanced interaction between robots only worked to form a specific lattice pattern which are known to the algorithm designer. In contrast, our algorithm accepts a description of the desired lattice as part of its input.

Much previous work addressed the formation problem in terms of the distributed task assignment problem. Smith and Bullo studied a geometric target assignment problem to let the robots move to the target autonomously and finally form a geometric formation [16]. However, their method relied on the assumption of an equal number of robots and target positions. Michael, Zavlanos, Kumar and Pappas proposed a market-based coordination algorithm in which the robots dynamically determine the formation pose by bidding for task assignment, if each robot knows the maximum number of robots that can be assigned to a certain task [12]. One limitation is that this algorithm does not guarantee the optimal assignment. Moreover, the task-assignment-based algorithms require the robots have knowledge of all the tasks in advance. Macdonald extended the work of Michael et al. [12] by assuming the robots acquire the tasks online. In his thesis the iterative closest point algorithm (ICP) is applied to the robots in order to match a set of robot positions to the set of model data of the target formations [10]. The author showed finally the target formations will be asymptotically stable using his improved method if each robot knows relative positions of all other robots, but the algorithm will not converge if some robots lose communication with other robots [10].

The task-assignment-based algorithms work well on solving multi-robot formation problems if the number of robots is not large. The major limitation is they require the robots to know tasks in the global sense. Therefore, they are expensive and difficult to apply directly to a large-scale multi-robot system.

Liu and Shell considered a special problem of formation control called "formation morphing", namely, the formation is changed as if it is gradually "deformed" in places, with the major pattern unaltered [9]. Their algorithm synthesized the graph matching problem and the assignment problem. The authors represented the robots routing trajectories using the Euclidean graph. These trajectories are projected from the augmenting paths in the bipartite graph so that the formation morphing can be achieved optimally [9].

Our algorithm distributes the assignment problem to each robot and we represent the desired formation using a graph

we call a lattice graph. Different from the Euclidean graph in Liu and Shell's work [9], we assign rigid the body transformation to the edges so that the lattice graph reflects both location and orientation relationship between two vertices. By virtue of the lattice graph notation, we can run our algorithm to generate various lattice formations more than repeated line, triangle, square, hexagon lattice patterns with minimal cost given a set of robots.

III. Problem Statement

This section defines the lattice formation problem we address in this paper.

A. Robots

We consider a collection of n identical **robots** $R = \{r_1, \ldots, r_n\}$ moving through an obstacle-free plane.

1) Robot identification numbers: Each robot $r \in R$ is assigned a unique **identification (ID) number** $\mathrm{id}(r)$. These IDs can be assigned arbitrarily, as long as they are selected from a total order whose binary relation we denote as $<$. Given the IDs $\mathrm{id}(r_i)$ and $\mathrm{id}(r_j)$ of two robots, a robot can decide whether $\mathrm{id}(r_i) < \mathrm{id}(r_j)$ or $\mathrm{id}(r_j) < \mathrm{id}(r_i)$. In practice, such IDs could be assigned, for example, based on the serial number or network MAC address of each robot.

2) Poses and coordinate frames: The **pose** $p_i = (x_i, y_i, \theta_i)$ of each robot r_i consists of its position and its orientation, expressed in an arbitrary but fixed global coordinate frame. At a certain time t, write the pose of r_i as $p_i(t)$. In addition to this global frame, we also define a **body frame** attached to each robot, in which the robot is always at the origin, facing along the positive x-axis. Let $T(r_i)$ denote the 3×3 homogeneous matrix representing the rigid body transformation from the body frame of r_i to the global frame:

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

Note that we use this global frame for modeling purposes only; the robots do not, in general, know their own global coordinates. Thus, we denote the local pose of robot r_j in the body frame of robot r_i using $p_j^{(i)} = (x_j^{(i)}, y_j^{(i)}, \theta_j^{(i)})$.

3) Motion: Each robot can rotate in place with maximum angular velocity $\omega_{\max}$ or move forward with maximum linear velocity $v_{\max}$.

4) Sensing and communication: Each robot can sense and communicate with other robots that are within a short distance, called the robots' **range** and denoted ϕ, of its own position. The other robots within that range are called the **neighbors** of that robot. Specifically, every Δt seconds:

- The sensors of each robot r_i generate a collection of **observations**, indicating the IDs and relative poses (that is, with coordinates expressed in the body frame of r_i) each of its neighbors.
- The communication hardware of each robot r_i can broadcast a short **message** to its neighbors. Details about the specific messages used in our algorithm appear in Section IV.

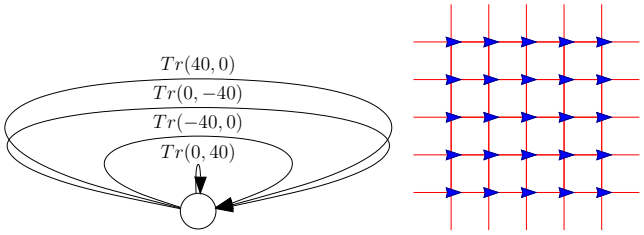


Fig. 2. A lattice graph in (a) creates a repeated square pattern. There is one node with four outgoing edges connecting to itself. Each edge is associated with a translation $Tr(x, y)$ of position (x, y) relative to the current node. Here the edge length is 40. For a robot playing the role of this node, its neighbors should be in locations $(40, 0)$, $(0, -40)$, $(-40, 0)$ and $(0, 40)$ to form the repeated square pattern in (b).

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

B. Lattice graphs

We use a directed graph to represent the desired pattern that the robots should form.

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

Figure 2 shows an example lattice graph. The intuition is that, when the lattice is fully formed, 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.

*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 structure of G. Specifically, for any i and j, if r_i and r_j are neighbors, there must exist an edge $e_{ij} : f(r_i) \longrightarrow f(r_j)$ in E, such that*

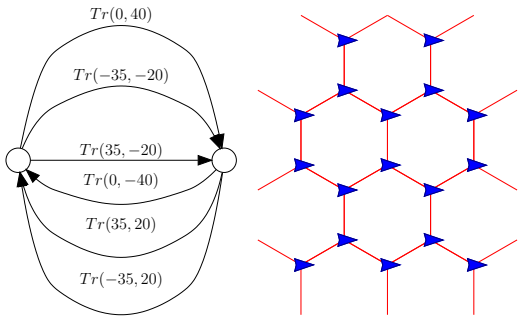
$$T(r_j) = T(r_i)T(e_{ij}). \tag{2}$$

That is, we require that $T(e_{ij})$ describes the relative pose of r_j in the body frame of r_i.

Informally, we want to position the robots so that each robot can be mapped to a vertex in the lattice graph called its **role** and the relative poses of each of robot's neighbors match that transformations that label the out-edges of its role vertex.

C. Evaluation criteria

To measure how well our algorithm works, we use two evaluation criteria.



(a) (b)

Fig. 3. A lattice graph in (a) creates a repeated hexagon pattern. There are two nodes, each has three outgoing edges connecting to the other. Each edge is associated with a translation of pose relative to the current node. Here the edge length is 40. For a robot playing the role of the vertex on the left side, its neighbors should be in locations $(0, 40), (-35, -20)$ and $(35, 20)$; for a robot playing the role of the vertex on the right side, its neighbors should be in locations $(0, -40), (35, 20)$ and $(-35, 20)$ to form the repeated hexagon pattern in (b).

1) Execution time: First, we are interested in the amount of time needed for the robots to reach a static state.

*Definition 3: A robot r_i is **static** at time t if its pose p_i remains the same at all future times, so that*

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

We define the **execution time** $\bar{t}_{exe}$ of the system as the smallest time at which all of the robots are in static states:

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

Our goal is to keep $\bar{t}_{exe}$ as small as possible.

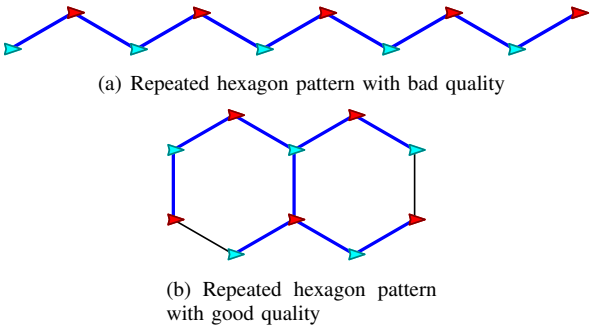
2) Formation Non-fulfillment Ratio: In addition to the execution time, we are also interested in the quality of the final formation.

In our algorithm, the robots continue moving until they reach a set of poses that satisfies the lattice graph, with each robot keeping track of its role with that graph. For a given robot r_i, let N_i denote the number of neighbors of r_i when our algorithm completes, and let E_i denotes the number of outgoing edges of its role vertex. Then we can measure the overall lattice quality using a non-fulfillment ratio:

$$\Gamma = \frac{1}{n} \sum_{i=1}^{n} \frac{E_i - N_i}{E_i} \tag{3}$$

Informally, Γ measures the average fraction a robot's potential neighbors that are "missing" in the final formation. Because, in a stable formation, each robot r_i has $0 \leq N_i \leq E_i$, possible values for Γ range from 0 to 1. We prefer smaller values for Γ.

Figure 4(a) and Figure 4(b) show two examples, in which both sets of robots satisfy the lattice graph in Figure 3(a) but with different non-fulfillment ratios.



(a) Repeated hexagon pattern with bad quality

(b) Repeated hexagon pattern with good quality

Fig. 4. To form repeated hexagon pattern with 10 robots, final formation can vary due to the initial configurations. (a): formation non-fulfillment ratio $\Gamma = 0.4$ (b): formation non-fulfillment ratio $\Gamma = 0.267$.

IV. Algorithm Description

This section introduces our algorithm. The basic idea is that each robot repeatedly accepts incoming messages, uses those messages to decide which other robot, if any, it should accept a task assignment from that one. Each robot computes and broadcasts a new task assignment for its own neighbors, and finally moves toward the destination specified its assigned task. Throughout this process, the robots exchange messages. Each message contains an authority (defined in Section IV-A) and a matching (defined in Section IV-B).

A. Authorities

The key idea to make our local strategy reach a global static state is the **authority** carried by each robot. This section defines authorities, provides a comparison operation for them, and shows how the robots will use authorities to form tree structures.

1) Authority definition:

*Definition 4: An **authority** is an ordered list of robot IDs*

$$\Lambda = \langle \mathrm{id}_1, \ldots, \mathrm{id}_k \rangle.$$

*The first ID in the list, id_1 is called the **root** ID. Likewise, the final ID in the list, id_k is called the **sender** ID. The number k of IDs in the list is called its **length**.*

The intuition is that, as messages are passed through the system, the authorities stored in those messages will identify a chain of robots that have agreed to accept task assignments from the previous robot, starting from the sender id_k and continuing back to the root id_1. This definition includes the possibility that $k = 1$, in which the root and sender are the same robot.

Because our algorithm depends on each robot identifying the highest authority in its neighborhood, the primary operation we need for authorities is to compare two authorities.

*Definition 5: Given two authorities $\Lambda_1 = \langle \mathrm{id}_1^1, \ldots, \mathrm{id}_k^1 \rangle$ and $\Lambda_2 = \langle \mathrm{id}_1^2, \ldots, \mathrm{id}_l^2 \rangle$, Λ_2 is **higher than** Λ_1 if*

1) $\mathrm{id}_1^2 > \mathrm{id}_1^1$, *or*
2) $l < k$ *if* $\mathrm{id}_1^2 = \mathrm{id}_1^1$, *or*
3) $\mathrm{id}_l^2 > \mathrm{id}_k^1$, *if* $\mathrm{id}_1^2 = \mathrm{id}_1^1$ *and* $l = k$.

This definition gives first priority to authorities whose roots have higher IDs, second priority to authorities that require fewer steps back to the root, and finally prefers authorities whose sender robot has the highest ID. Notice that, under this definition some pairs of authorities are incomparable, namely, pair of authorities that have the same

root, sender, and length, but differ at some other position within the list. In the algorithm introduced below, this case does not occur, because in every set of authority comparisons we make, the sender elements will be unique.

2) Constructing authority trees: The robots use these authorities to establish a collection of authority trees. The idea is that each robot must decide which of its neighbors, if any, to select as its parent. Robots that choose not to select a parent are called **root** robots; robots that select a parent are called **descendants** of that parent.

To select a parent, robot r_i examines the messages $m_1, \ldots, m_j$ it has received from its neighbors in the most recent time step. Each message m contains an authority which we denote $\Lambda(m)$.

First, r_i discards any messages m in which its own ID appears in $\Lambda(m)$. This step is important to prevent "authority cycles," in which the authority lists become arbitrarily long by repeating IDs, from occurring.

Next r_i forms an authority containing only its own ID, $\Lambda_i = \langle \mathrm{id}(r_i) \rangle$ and compares it to the authorities of the remaining messages, selecting the highest from this set. Let $\hat{\Lambda}$ denote the highest such authority.

Using this authority, r_i can decide whether to become a root or a descendant.

- If $\hat{\Lambda} = \Lambda_i$, then r_i decides to be a root at this time step. In the message that it broadcasts in this time step, it uses Λ_i as the authority.
- Otherwise, if $\hat{\Lambda} \neq \Lambda_i$, then r_i selects the sender robot of the message that contained $\hat{\Lambda}$ as its parent. To create the authority for r_i to use in its message at this time step, we append $\mathrm{id}(r_i)$ to the end of $\hat{\Lambda}$.

Notice that, in the first step, before any messages are transmitted, every will consider itself a root. As messages are transmitted, more and more robots will attach themselves to a tree structure, in which each tree is rooted at the highest-ID robot in each connected component of the communication graph.

B. Local task assignment

After selecting its parent or choosing to be a root, each robot then uses a task assignment algorithm to decide, based on its observations, an optimal matching of its neighbors with potential destinations in its body frame. This concept is closely related to the idea of roles in the lattice graph, because the correct neighbor poses depend on the edges outgoing from each robot's role vertex.

This definition explains the specific type of task assignment that each robot computes.

Definition 6: Given a robot r_i and a role vertex v_i for that robot, let the lattice graph edge set $L = \{\emptyset, e_{ij}, e_{ik}, \ldots\}$ be the set that contains a null value $\emptyset$ and all outgoing edges from vertex v_i. Let $Q = \{\mathrm{id}(r_a), \mathrm{id}(r_b), \ldots\}$ be the set that contains the IDs of the neighbors of r_i. Then a ***matching for*** *r_i is a function $\eta : Q \to L$ that associates each neighbor ID with either a lattice graph edge from its role vertex or with the null value.*

The process of computing these matchings differs depending on whether the robot is a root or a descendant.

1) Root matchings: Consider a root robot r_i with N_i neighbors around it. By convention, each root robot selects the first lattice graph vertex as its role, $f(r_i) = v_0$. Let E_i denote the number of outgoing edges of this role vertex.

The goal is to form a matching for r_i. This matching attempts to associate each of these N_i neighbors with one of these E_i edges. Notice that, if $N_i > E_i$, that is, if r_i has more neighbors than $f(r_i)$ has out-edges, then some neighbors cannot be assigned to edges, and instead must be matched with $\emptyset$. The intuition is that if a neighbor r_j is associated with with an edge e, then r_j should move so that the transform between its body frame and the body frame of r_i is equal to $T(e)$. Informally, r_j should "follow" the edge. If a neighbor r_j is associated with the null value, r_j is said to have "no match," and should travel to a different part of the formation. (Section IV-D describes the details.)

To compute a matching, we define a weight matrix of size $N_i \times \max(N_i, E_i)$.

1) Each row of the matrix corresponds to a neighbor of r_i;
2) Each column of the matrix corresponds to an out-edge of $f(r_i)$ or to a copy of the null value $\emptyset$.
3) The entries of this matrix in columns $1, \ldots, E_i$ represent the Euclidean distance between the current position of each neighbor (determined by the observations of r_i) and its desired position if matched with a given edge (determined by the transformation of the edge), both computed in the body frame of r_i. Specifically, to find the desired position $(\bar{x}^{(i)}, \bar{y}^{(j)})$ for a given edge e, we can use its transformation $T(e)$:
$$[\bar{x}^{(i)} \quad \bar{y}^{(i)} \quad 1]^\top = T(e)[0 \quad 0 \quad 1]^\top \tag{4}$$
4) Columns beyond E_i, which correspond to the null value instead of edges, have their entries set to ∞. The intuition is that assigning a neighbor to the null value is a "last resort" that should not be selected if there are any finite alternatives.

Based on this matrix, we apply Kuhn's Hungarian algorithm [6] to compute the optimal matching. This algorithm takes $O(N_i^3)$ running time to generate the matching with the minimum total weight. The idea is that this matching represents r_i's best idea of how to form the desired lattice locally. The resulting matching is transmitted, along with the authority discussed in Section IV-A, to the robot's neighbors in the next time cycle.

Figure 5 shows an example this matching process, using the lattice graph in Figure 2. The center robot, which is a root, assigns four of its neighbors to desired poses in its body frame and lets the other robot move out of its range.

2) Descendant matchings: The process for computing matchings when r_i is a descendant robot is similar, but has two vital differences. In the following paragraphs, we use r_p to denote the parent of r_i.

First, a descendant robot chooses its role according to the matching received from r_p. Specifically, the role of r_i is

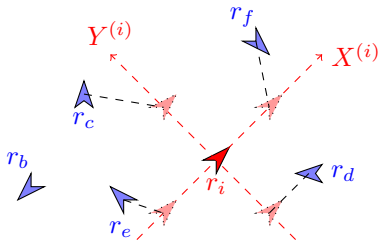


Fig. 5. Robot r_i is a root and assigns its neighbors r_b, r_c, r_d, r_e, r_f four desired destinations computed from lattice graph edges, respectively. Robot r_b is not assigned to any destination by r_i so it will move away from r_i.

the terminal vertex of the edge associated with its own ID in that matching. The coincides with the intuition that r_i should follow its parent's instructions.

Second, the matching from r_p places some constraints on the matching that r_i should generate. Specifically, r_i should ensure that r_p is matched with an edge whose transformation is the inverse of the edge matched to r_i by r_p. To enforce this constraint, we modify the weight matrix to have ∞ entries throughout the row for r_p and the column for this inverse edge, except where that row and column meet, where we assign 0. Because of this modification, any optimal task assignment will match r_p to this edge, as desired.

Aside from these differences, descendant robots compute and broadcast their local matchings in the same way as root robots.

C. Robot motion

The final step of our algorithm is to decide where to move.

The case of a root robot is very simple: Root robots do not move. The reasoning is that, since the poses of the other robots are determined relative to some root, there is nothing to gain from a root robot changing its pose.

Selecting movements for descendant robots is somewhat more complex. We have already described how each descendant r_i selects a parent r_p which describes an ultimate destination $\bar{p}_i^{(p)}(t)$ for r_i, based on currently available information.

However, to ensure that it remains within range of its parent, r_i may need to choose an intermediate destination for the next time step that is not along the direct path toward $\bar{p}_i^{(p)}(t)$. This constraint is complicated by the fact that r_i does not know the motion that r_p plans to make in the next time step. The following lemma will be useful for staying within range of the parent, in spite of this uncertainty.

Lemma 7: If robot r_i is the neighbor of robot r_p at time t, it will still be the neighbor of r_p at time $t+\Delta t$ if $||p_i(t+\Delta t) - p_p(t)|| \leq \phi - v\Delta t$, where v is the velocity of r_i and r_p.

Proof: Because r_i and r_p are initially neighbors, we have

$$||p_i(t) - p_p(t)|| \leq \phi. \tag{5}$$

Due to the velocity limits on the robots, we also know that

$$||p_p(t+\Delta t) - p_p(t)|| \leq v\Delta t. \tag{6}$$

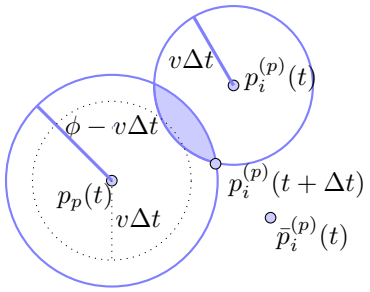


Fig. 6. The actual pose $p_p(t+\Delta t)$ of r_p could be anywhere in the dotted circle. The desired pose $\bar{p}_i^{(p)}(t)$ for r_i is assigned by r_p at time t. On the boundary of the intersection of $P_p \cap P_i$ (shaded area), we choose the nearest point to the desired ultimate destination position as the real destination position for r_i at time $t+\Delta t$.

since we assume that

$$||p_p(t) - p_i(t+\Delta t)|| \leq \phi - v\Delta t \tag{7}$$

then applying triangle inequality to Equation 6 and Equation 7 yields

$$\begin{aligned} ||p_p(t+\Delta t) - p_i(t+\Delta t)|| &\leq ||p_p(t+\Delta t) - p_i(t)|| \\ &\quad + ||p_i(t) - p_i(t+\Delta t)|| \\ &\leq \phi - v\Delta t + v\Delta t = \phi \end{aligned}$$

□

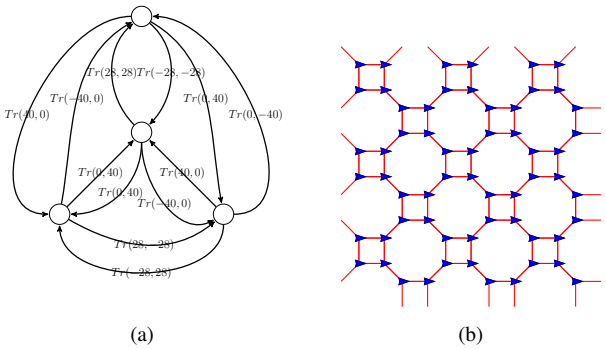
Lemma 7 shows that, to ensure that r_i remains within range of r_p, it must move to a position with distance at most $\phi - v\Delta t$ of r_p current position. However, r_i itself also has limited velocity. Therefore, it must choose a position within distance $v\Delta t$ of its own current position. These constraints lead us to the following approach, which is illustrated in Figure 6.

1) First, r_i computes the desired ultimate destination pose $\bar{p}_i^{(p)}(t)$, as shown in Equation 8.
2) Next, r_i computes the set P_i of reachable positions of itself at $t+\Delta t$ using its position as center and radius $v\Delta t$ and the set P_p of reachable positions of r_p at $t+\Delta t$ using r_p's position as center and radius $\phi - v\Delta t$. Both of these sets disc-shaped subsets of the plane.
3) Finally, r_i chooses for its intermediate destination, $p_i^{(p)}(t+\Delta t)$, the nearest point to $\bar{p}_i^{(p)}(t)$ in the intersection of P_i and P_p.

After choosing its destination in this way, r_i begins to navigate toward $p_i^{(p)}(t+\Delta t)$ in a straightforward way: It turns itself to face the target position and directly drives itself there.

As a final step, if r_i reaches its ultimate destination position, it must also achieve the orientation specified by its edge's rigid body transformation. The required orientation can be computed by transforming a point on the x-axis of r_i's body frame into the body frame of r_p:

$$\begin{aligned} \begin{bmatrix} \cos\bar{\theta}_i^{(p)} & \sin\bar{\theta}_i^{(p)} & 1 \end{bmatrix}^\top &= T(e) \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^\top \\ &\quad - \begin{bmatrix} \bar{x}_i^{(p)} & \bar{y}_i^{(p)} & 1 \end{bmatrix}^\top \end{aligned} \tag{8}$$



(a) (b)

Fig. 7. A lattice graph in (a) creates a repeated octagon-square pattern. There are four nodes, each has three outgoing edges connecting to the other three nodes. Each edge is associated with a translation of pose relative to the current node. The edge length is 38. The repeated octagon-square pattern is shown in (b).

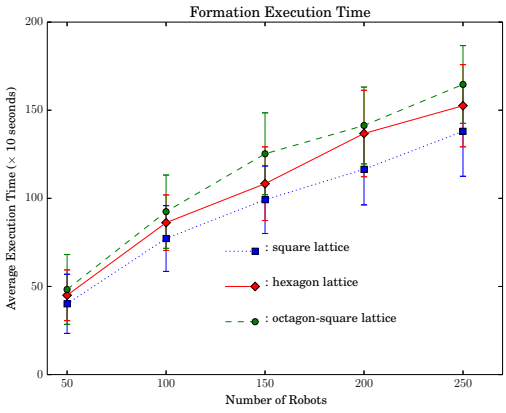


Fig. 8. Average execution time and its standard deviation of forming repeated lattice patterns of square, hexagon and octagon-square.

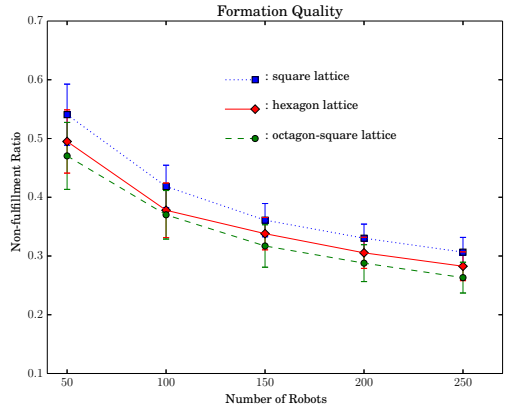


Fig. 9. Average non-fulfillment ratio and its standard deviation of forming lattice patterns of square, hexagon and octagon-square.

D. Unmatched robots

One final algorithmic detail is to explain how a robot should move when it is matched to the null value $\emptyset$, instead of matching a lattice graph edge of its parent. Section IV-B.1 mentioned that such robots should "travel to a different part of the formation."

Specifically, if a descendant robot r_i receives a message from its parent, in which it is matched to $\emptyset$, r_i concludes that its current location is too congested with robots. In response, r_i moves directly away from its parent for a fixed period of time, equal to $2\phi/v$. During this time, r_i does not communicate with anyone, and is neither root nor descendant. When this time expires, the robot resumes normal execution. The effect is to disperse the robots without the oscillations that could occur of r_i selects a new parent immediately afterward.

V. Implementation and Results

We implemented a simulation of this algorithm in C++, using Liu's implementation of the Hungarian algorithm [7] [8] [9]. We designed experiments to verify the scalability and efficiency of our algorithm by measuring the qualities of different lattice formations using Equation 3 and measuring the execution time $\bar{t}_{exe}$ given different number of robots.

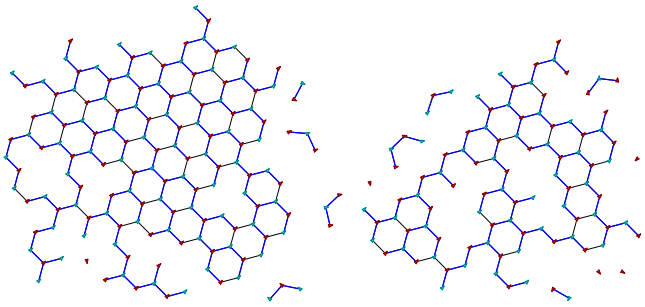
First, we conducted experiments to let robots form three kinds of repeated lattice patterns: square (Figure 2), hexagon (Figure 3) and octagon-square (Figure 7) in an obstacle-free environment. For each pattern, we performed a series of experiments. We keep a consistent velocity in all experiments. We varied the number of robots n between 50 and 250 in increments of 50. For each n, we repeated the experiment 50 times with uniform distributions of initial poses randomly generated given distinct random seeds, and computed the execution time $\bar{t}_{exe}$ and the average non-fulfillment ratio Γ when they reach static states. Figure 8 and Figure 9 plot the results.

Figure 8 shows that the execution time for robots to reach static states increases increases linearly with the number of robots, across all three given lattice graphs.

The final formation non-fulfillment ratio can be impacted by the distribution of the initial poses. Figure 9 shows that when the number of robots is small and robots are sparsely distributed initially, most of robots do not have enough neighbors around thus the average formation non-fulfillment ratio is larger than the scenario where the distribution of robots' poses is dense.

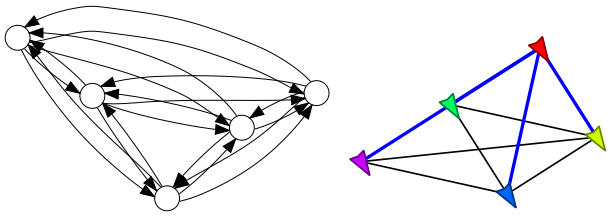
The average non-fulfillment ratio for all three lattice patterns decreases, although rather slowly, when increasing the number of robots. Ideally, we expect the non-fulfillment ratio for repeated lattice formation to converge to zero as the number of robots increases. Figure 9 shows this trend slightly. Note that the more robots who move to some places where that few neighbors, the higher the non-fulfillment ratio will be. In other words, although the number of robots increases, the number of those who are finally dispersed also increases.

Next, we conducted experiments to verify the robustness of our algorithm. Figure 10 shows one test example in which we removed some robots from the system when the system reached the static state at the first time. Initially we let 150



(a) (b)

Fig. 10. Figure (a) shows the formation of repeated hexagon lattice with 150 robots. Figure (b) shows the reconfigured formation after randomly removing 50 robots.



(a) (b)

Fig. 11. Figure (a) shows the lattice graph with 5 nodes and 20 edges. We omitted showing the translations of the edges. Figure (b) shows the simulation result of the final formation. Each thick line connects a descendant robot to its parent.

robots to form a repeated hexagon pattern using lattice graph shown in Figure 3(a). It took 144.08 execution time for all robots to reach static states, with the non-fulfillment ratio $\Gamma = 0.196$. After randomly removing 50 robots (simulating a massive failure), it took another 168.52 execution time to reach static states again with the non-fulfillment ratio $\Gamma = 0.287$. Similar experiments showed that our algorithm also works if adding more robots to the system.

Finally, our algorithm can also be used to form non-repeating lattice patterns. Figure 11 shows a simple case in which five robots formed the letter "F," given a complete five-node lattice graph. This special case is similar to the problem addressed by Michael et al. [12] and Macdonald [10]. It requires all robots be able to observe and communicate with each other initially.

VI. Conclusion and Future Work

The experimental results from conducted simulations demonstrate the effectiveness of our algorithm. Also this decentralized algorithm scales reasonably well with increasing numbers of robots, and is robust to the situation when some robots are removed from or with more robots added to the system.

Currently, our algorithm does not use a collision avoidance mechanism as part of the robots' motion strategy. In our upcoming implementation with physical robots, we will consider let the robot pass around the other robots on its way.

Moreover, we plan to improve the motion strategy for the situation when a robot needs to leave from its parent who does not assign it a destination, so that better lattice formation quality can be achieved.

Another major limitation of our work is that the algorithm can only be applied to the holonomic robots and differential drive vehicles. It would be interesting to extend this method for the car-like robot systems as our future work.

Acknowledgments This material is based upon work supported by the National Science Foundation under Grant No. IIS-0953503.

References

[1] E. Bahceci, O. Soysal, and E. Sahin, "A review: Pattern formation and adaptation in multi-robot systems," *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-03-43*, 2003.

[2] Y. U. Cao, A. S. Fukunaga, A. B. Kahng, and F. Meng, "Cooperative mobile robotics: antecedents and directions," in *Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, vol. 1, 1995, pp. 226–234 vol.1.

[3] G. Dudek, M. R. M. Jenkin, E. Milio, and D. Wilkes, "A taxonomy for multi-agent robotics," *Autonomous Robots*, vol. 3, no. 4, pp. 375–397, 1996. [Online]. Available: http://dx.doi.org/10.1007/BF00240651

[4] 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.

[5] 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.

[6] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955. [Online]. Available: http://dx.doi.org/10.1002/nav.3800020109

[7] L. Liu and D. A. Shell, "Assessing optimal assignment under uncertainty: An interval-based algorithm," *Int. J. Rob. Res.*, vol. 30, no. 7, pp. 936–953, Jun. 2011. [Online]. Available: http://dx.doi.org/10.1177/0278364911404579

[8] ——, "Large-scale multi-robot task allocation via dynamic partitioning and distribution," *Autonomous Robots*, vol. 33, no. 3, pp. 291–307, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10514-012-9303-2

[9] ——, "Multi-robot formation morphing through a graph matching problem," in *International Symposium on Distributed Autonomous Robotic Systems (DARS)*, Baltimore, MD, Nov 2012.

[10] E. A. Macdonald, "Multi-robot assignment and formation control," Master's thesis, Georgia Institute of Technology, 2011.

[11] E. Martinson and D. Payton, *Lattice formation in mobile autonomous sensor arrays*. Springer Berlin Heidelberg, 2005, vol. 3342, ch. Lecture notes in computer science, pp. 98–111.

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

[13] 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, pp. 229–242.

[14] 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.

[15] S. Prabhu, W. Li, and J. McLurkin, "Hexagonal lattice formation in multi-robot systems," in *11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, 2012.

[16] S. L. Smith and F. Bullo, "A geometric assignment problem for robotic networks," in *Modeling, Estimation and Control: Festschrift in Honor of Giorgio Picci on the Occasion of his Sixty-Fifth Birthday*, A. Chiuso, A. Ferrante, and S. Pinzoni, Eds. Springer, 2007, vol. 364, pp. 271–284.

[17] W. M. Spears, R. Heil, and D. Zarzhitsky, "Articial physics for mobile robot formations," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC'05)*, 2005.

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