Distributed Formation of Arbitrary Lattice Pattern for Multi-robot Systems

Yang Song, Jason M. O'Kane
song24@email.sc.edu
jokane@cse.sc.edu

Dept. of Computer Science and Engineering
University of South Carolina

UNIVERSITY OF SOUTH CAROLINA

Contribution

- WHAT DO WE HAVE:
  Given a set of autonomous robots, each robot knows its local information.
- INPUT:
  Representations of various formation patterns.
- OUTPUT:
  Robot motion control.
- WHAT DO WE EXPECT:
  Various formation patterns.

Robot Specifications

- Holonomic or Differential Drive robots.
- Each robot has an unique **ID**.
- Each robot has three states, use a vector $p = [x, y, \theta]^T$ to represent its **pose**.
- Each robot has a **range** within which it can sense and communicate with other robots.
- Each robot gets **observation** of its neighbors' IDs and relative poses in its body frame.

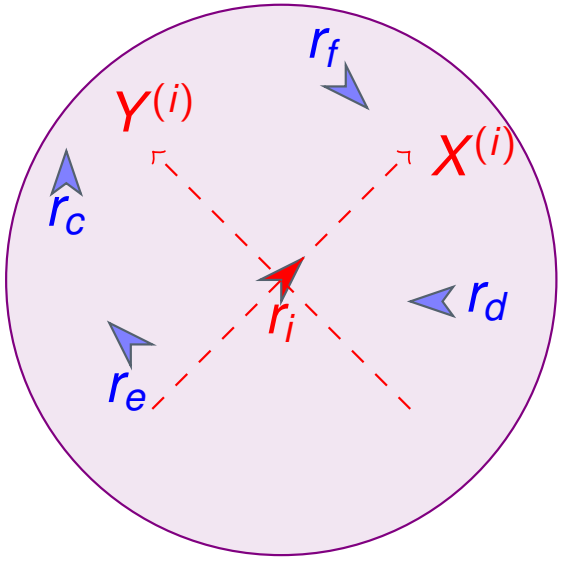


Figure: Robot r_i and its neighbors r_c, r_d, r_e, r_f in its local $X^{(i)}$-$Y^{(i)}$ coordinates. The disk describes r_i's range.

Lattice Graph

Definition

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 $w \xrightarrow{T(e)^{-1}} v$.

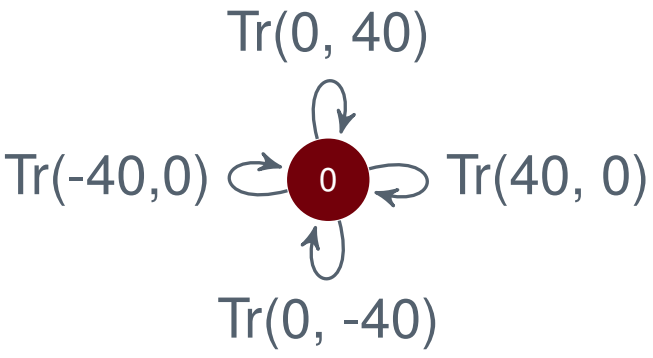
Given a lattice graph $G = (V, E)$ and a set of robots $R = \{r_1, \ldots, r_n\}$, we say that R **satisfies** G if there exists a role function $f : R \to V$ that preserves the neighborhood structure of G. Specifically, for any i and j, if r_i and r_j are neighbors, there must exist an edge $e_{ij} : f(r_i) \longrightarrow f(r_j)$ in E, such that

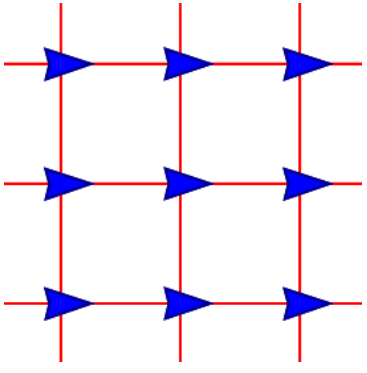
$$T(r_j) = T(r_i)T(e_{ij})$$

Lattice Graph

Example: lattice graph for repeated square patterns

- 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.
- 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 $(0, 40)$.





Authority

Definition

An **authority** is an ordered list of robot IDs

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

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**.

Each robot broadcasts messages to its neighbors.

A message contains:

- The robot's authority.
- The robot's matching.

Algorithm

General Algorithm Description

1. Each robot decides whether to be a root robot or a descendant robot based on “authority”.

Algorithm

General Algorithm Description

1. Each robot decides whether to be a root robot or a descendant robot based on “authority”.
2. Each robot chooses a role.
   - Root robots always select the first lattice vertex as their role.
   - Descendant robots accept their roles as from their parents.

Algorithm

General Algorithm Description

1. Each robot decides whether to be a root robot or a descendant robot based on “authority”.
2. Each robot chooses a role.
   - Root robots always select the first lattice vertex as their role.
   - Descendant robots accept their roles as from their parents.
3. Each robot computes a local task assignment for its neighbors and broadcasts this assignment to its neighbors.

Algorithm

General Algorithm Description

1. Each robot decides whether to be a root robot or a descendant robot based on “authority”.
2. Each robot chooses a role.
   - Root robots always select the first lattice vertex as their role.
   - Descendant robots accept their roles as from their parents.
3. Each robot computes a local task assignment for its neighbors and broadcasts this assignment to its neighbors.
4. Each descendant robot moves toward to position assigned by its parent.

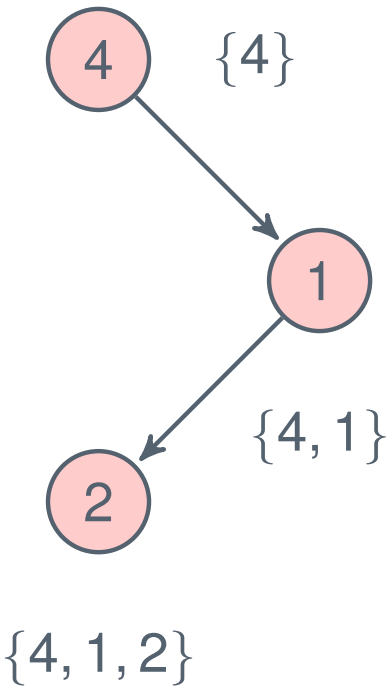
Construct Authority Tree

1. Decide to be root or descendant

The key idea to make our local strategy reach a global static state is the authority carried by each robot

- Each robot chooses one of its neighbors to be its parent.
- Robots that choose not to select a parent are called **root** robot.
- Robots that select a parent are called **descendants** of that parent.

Robot 4 is the Root robot
Robot 1 is a descendant of Robot 4
Robot 2 is a descendant of Robot 1



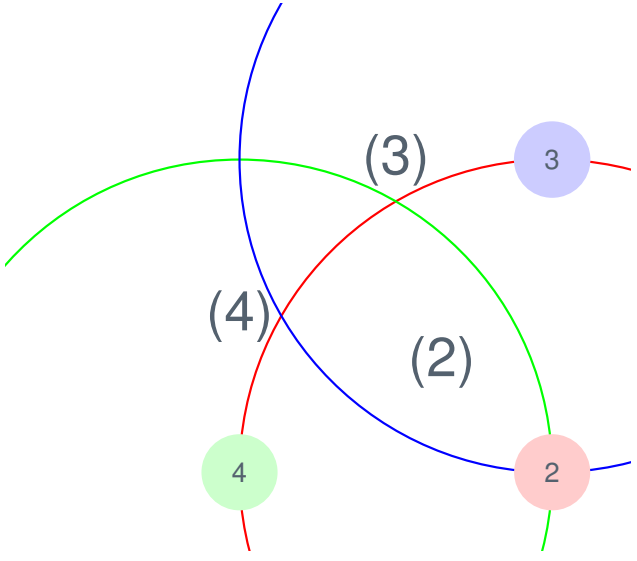
Construct Authority Tree

1. Decide to be root or descendant

The robots use these authorities to establish a collection of authority trees

1. Discards any message in which the authority contains its own ID.
2. Forms an authority containing only its own ID, compares it with the authorities of remaining messages and selects the highest authority.

- If its authority is the highest, then it is the root;
- Otherwise, it selects the one who sends the highest authority as its parent. Append its own ID to the highest authority to create its own authority.

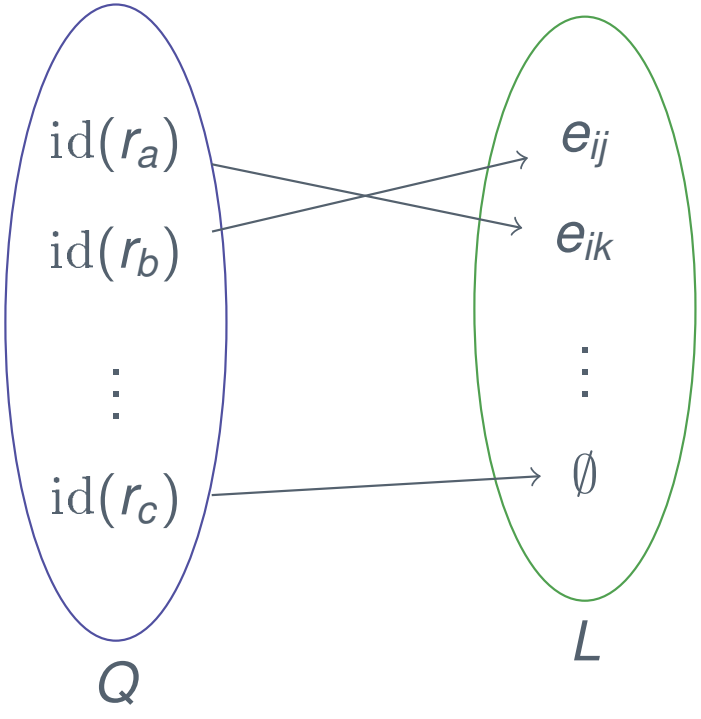


Construct Matching

2. Role selection

Given a robot r_i and a role vertex v_i for that robot, let the lattice graph edge set $L = \{\emptyset, e_{ij}, e_{ik}, ...\}$ be the set that contains a null value $\emptyset$ and all outgoing edges from vertex v_i. Let $Q = \{\mathrm{id}(r_a), \mathrm{id}(r_b), ...\}$ be the set that contains the IDs of the neighbors of r_i.

A **matching** for a robot is a function $\eta : Q \rightarrow L$ that associates each neighbor ID with either a lattice graph edge from its role vertex or with the null value $\emptyset$.



Local Task Assignment

3. Hungarain Algorithm

To compute an optimal matching of a robot with N neighbors and E out-going edges of its role in the lattice graph, define a weight matrix of size $N \times \max(N, E)$ and apply Hungarian Algorithm.

1. Each row corresponds to a neighbor;
2. Each column corresponds to an out-going edge of robot's role or a null value $\emptyset$.
3. The entries of the matrix is the Euclidean distance between current position of each neighbor and the desired position if matched with a lattice graph edge.

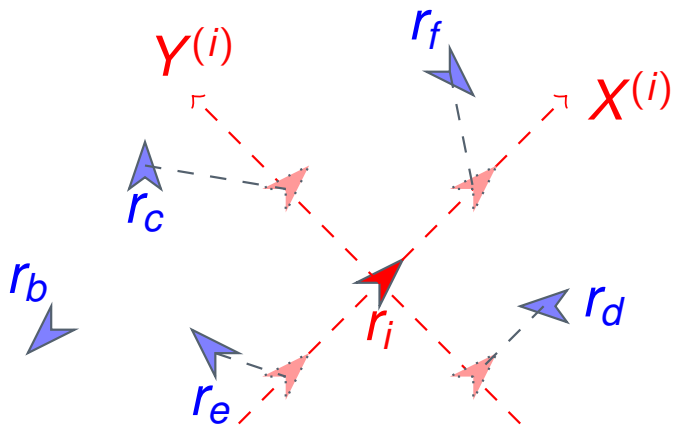
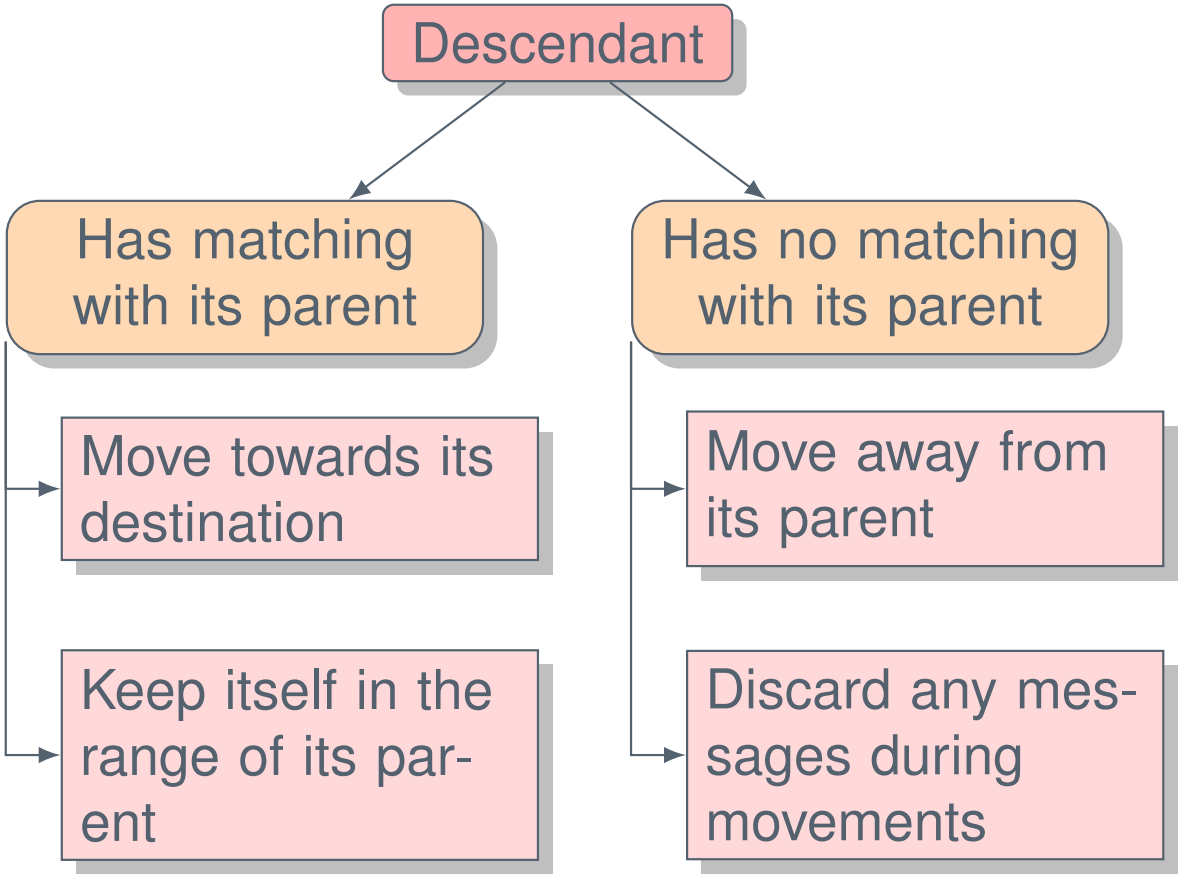


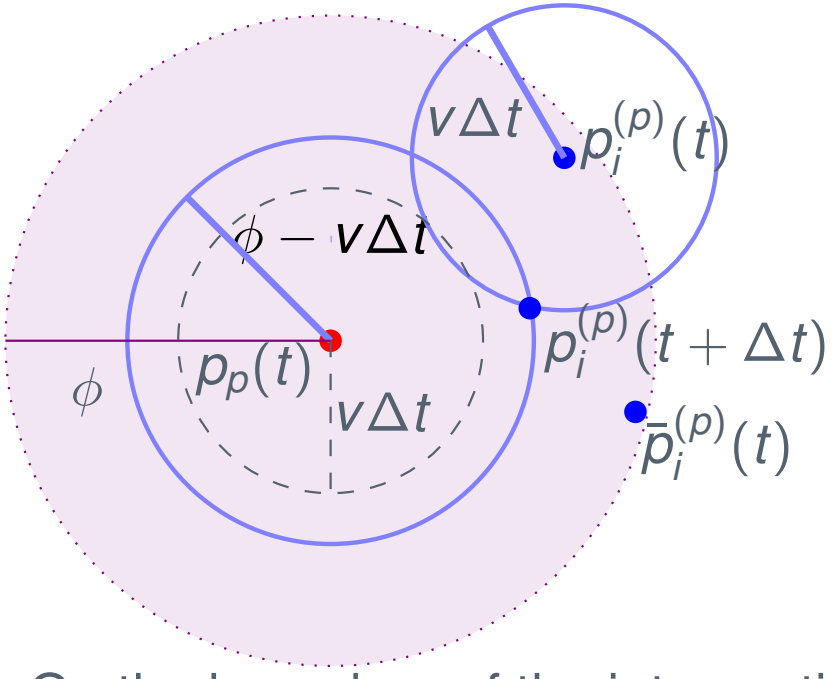
Figure: Robot r_i has its neighbors r_c, r_d, r_e, r_f match to one of its out-going edges, has r_b match to $\emptyset$.

1. Root Robot: Does Not Move
2. Descendant Robot:



Bounded Movement Example

At time t, robot r_i selects robot r_p as its parent.



- Shadowed region is the range of robot r_p at time t.
- The actual pose $p_p(t+\Delta t)$ of r_p could be anywhere in the dashed circle.
- The desired pose $\bar{p}_i^{(p)}(t)$ for robot r_i is assigned by r_p at time t.

On the boundary of the intersection of $P_p \cap P_i$, r_i choose the nearest point to the desired ultimate destination position as its real destination position at time $t + \Delta t$

Simulation

Octagon-square formation with 100 robots

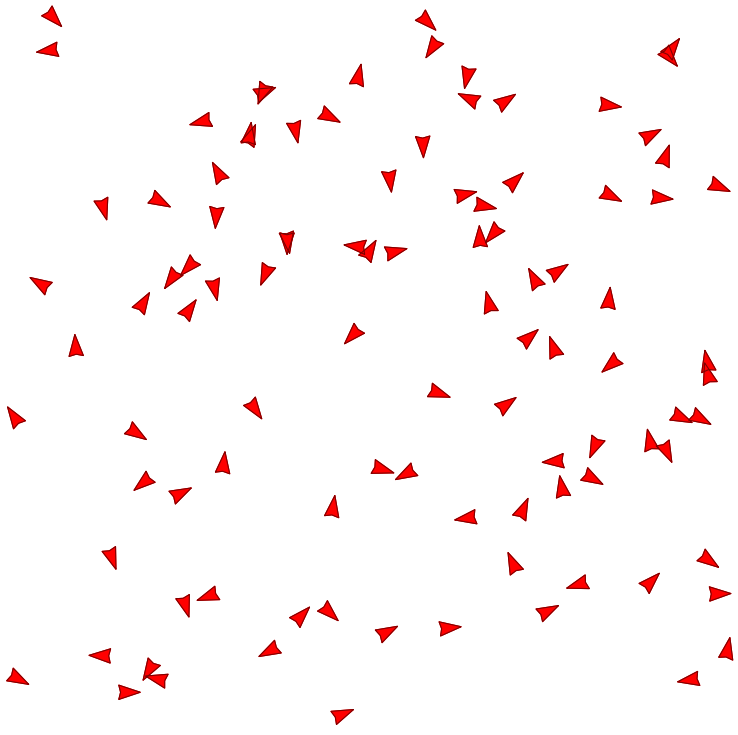


Figure: Initial uniform distribution of 100 robots's poses.

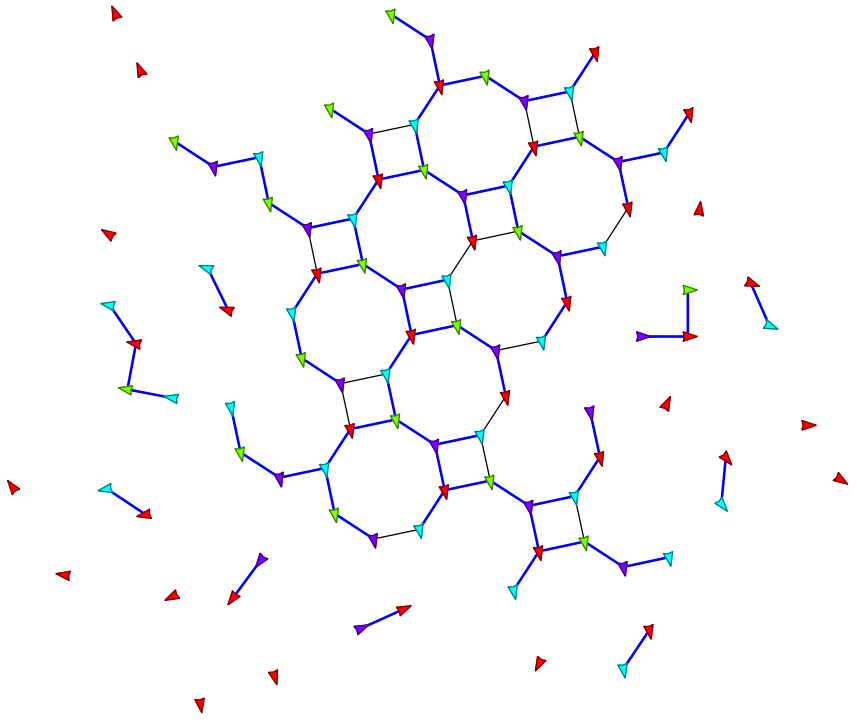


Figure: Final formation of repeated octagon-square pattern.

Simulation

Hexagon formation with 100 robots

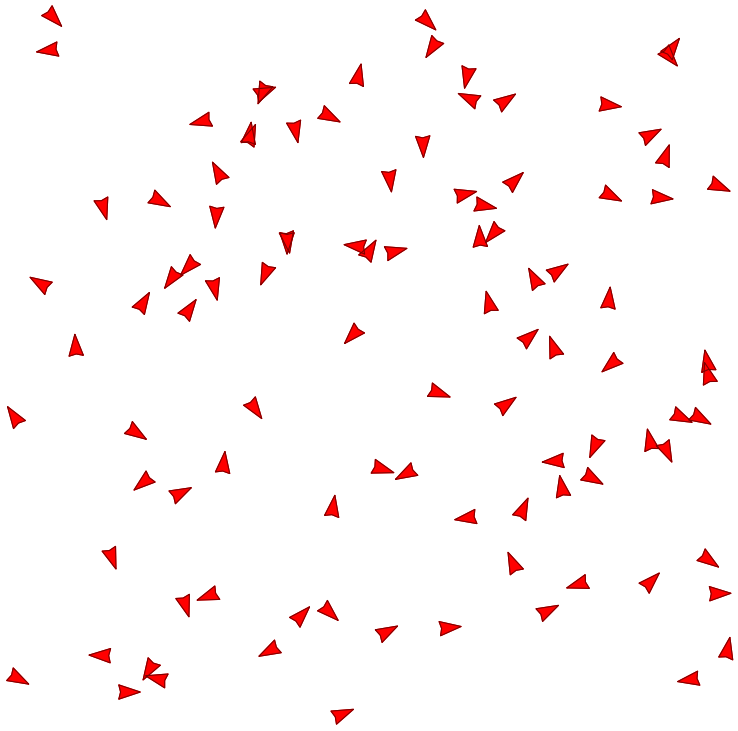


Figure: Initial uniform distribution of 100 robots's poses.

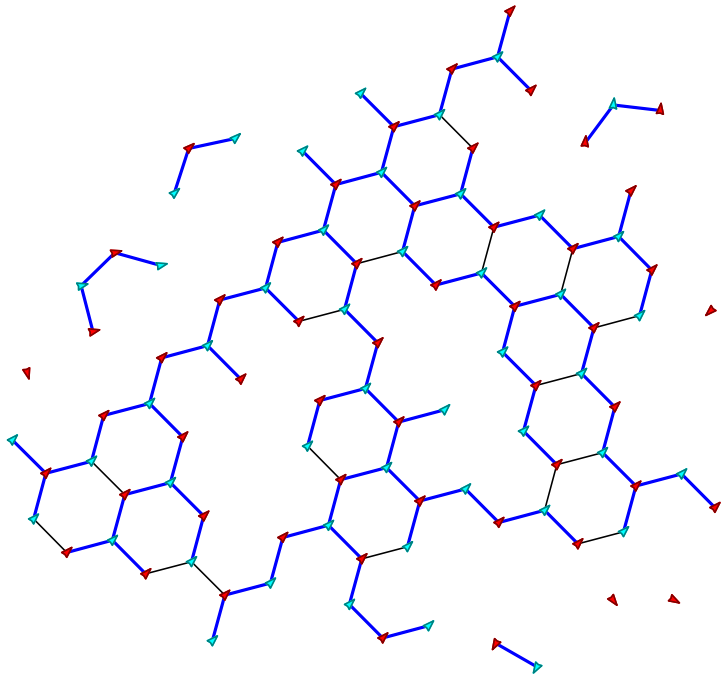


Figure: Final formation of repeated hexagon pattern.

Experiments
on three kinds of repeated lattice patterns

Experiments Setup

- Keep a consistent velocity in all experiments
- Varied the number of robots $n = 50, 100, 150, 200, 250$
- For each n, we repeated the experiment 50 times
- Initially all robots are uniformly distributed with distinct random seeds each experiment
- For robustness test, randomly remove some robots from or add some robots to the system

Measurements

- Average execution time
- Average non-fulfillment ratio: $\Gamma = \frac{1}{n} \sum_{i=1}^{n} \frac{E_i - N_i}{E_i}$

Results

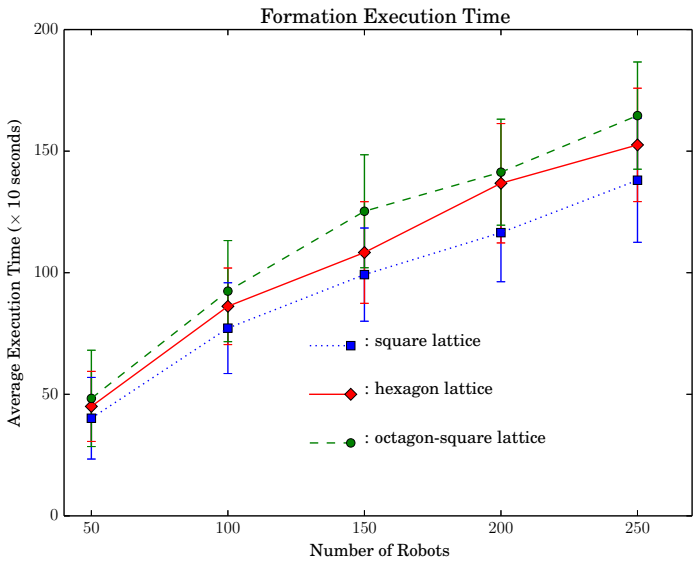


Figure: Average execution time and its standard deviation of forming repeated lattice patterns of square, hexagon and octagon-square

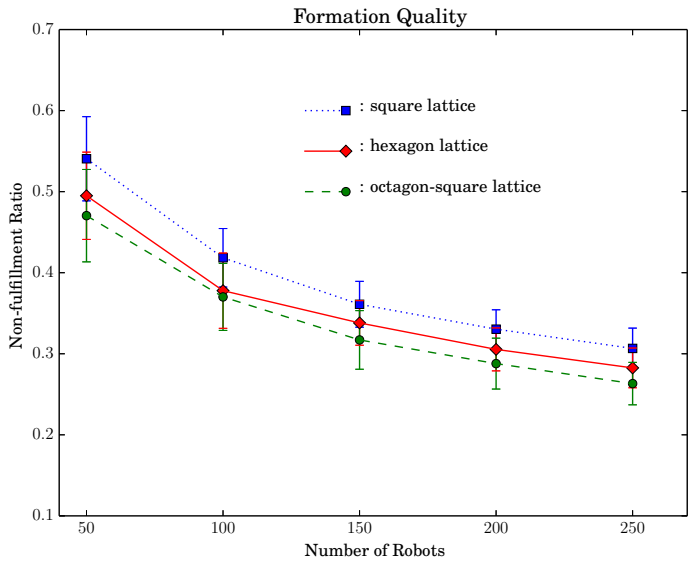


Figure: Average non-fulfillment ratio and its standard deviation of forming lattice patterns of square, hexagon and octagon-square.

Conclusions

Pros of the Algorithm

- robots can form different types of geometric formations, including the repeated lattice patterns
- scales reasonably well with increasing numbers of robot
- robust to the situation when some robots are removed from or with more robots added to the system

Cons of the Algorithm

- no collision avoidance for motion strategy
- limited on the multi-robot systems subject to the nonholonomic constraints

Questions

THANK YOU!