

RICE UNIVERSITY

Swarm Robotics: Measurement and Sorting

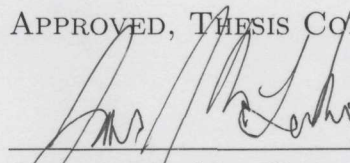
by

Yu Zhou

A THESIS SUBMITTED  
IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE

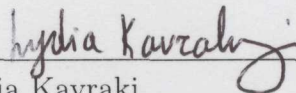
Master of Science

APPROVED, THESIS COMMITTEE:



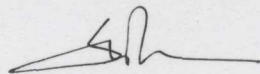
---

James McLurkin, Chair  
Assistant Professor of Computer Science



---

Lydia Kavraki  
Professor of Computer Science



---

Swarat Chaudhuri  
Assistant Professor of Computer Science

Houston, Texas

April, 2015

RICE UNIVERSITY

**Swarm Robotics: Measurement and Sorting**

by

**Yu Zhou**

A THESIS SUBMITTED  
IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE

**Master of Science**

APPROVED, THESIS COMMITTEE:

---

James McLurkin, Chair  
Assistant Professor of Computer Science

---

Lydia Kavraki  
Professor of Computer Science

---

Swarat Chaudhuri  
Assistant Professor of Computer Science

Houston, Texas

April, 2015

## ABSTRACT

Swarm Robotics: Measurement and Sorting

by

Yu Zhou

To measure is an important ability for robots to sense the environment and nearby robots. Although camera, laser, and ultrasonic provide very accurate measurements, they are expensive and not scalable for large swarm of low-cost robots. The r-one robot designed at Rice University is equipped with infrared transmitters and receivers, which are designed for remote control and are very inexpensive in mass production. They are a good solution for short-range communication, since the signal attenuates at about 1 to 2 meters with appropriate voltage. This work describes my results in using them to measure bearing, orientation, and distance between nearby robots. However, infrared receivers are not designed for this kind of use, so I present a variable transmit power approach to allow useful and efficient local geometry measurements.

With the ability to measure bearing and distance, I am able to solve the problem of sorting a group of  $n$  robots in a two-dimensional space. I want to organize robots into a sorted and equally-spaced path between the robots with lowest and highest label, while maintaining a connected communication network throughout the process. I begin with a straightforward geometry-based version of sorting algorithm, and point out there are many difficulties when communication range becomes limited.

Then I describe a topology-based distributed algorithm for this task. I introduce operations to break the symmetry between minimum and maximum, in order to keep

time, travel distance, and communication costs low without using central control. I run a set of algorithms (leader election, tree formation, path formation, path modification, and geometric straightening) in parallel. I show that my overall approach is safe, correct, and efficient. It is robust to population changes, network connectivity changes, and sensor errors. I validate my theoretical results with simulation results. My algorithm implementation uses communication messages of fixed size and constant memory on each robot, and is a practical solution for large populations of low-cost robots.

# Contents

|   |           |
|---|-----------|
| Abstract  | ii        |
| List of Illustrations   | vi        |
| List of Tables  | x         |
| <b>1 Introduction</b>   | <b>1</b>  |
| <b>2 Related Work</b>   | <b>4</b>  |
| <b>3 Model and Assumptions</b>                                    | <b>8</b>  |
| <b>4 Local Geometry Measurement</b>                               | <b>11</b> |
| 4.1 The r-one Robot . . . . .                                     | 11        |
| 4.2 Bearing and Orientation Measurement . . . . .                 | 13        |
| 4.3 Distance Measurement . . . . .                                | 19        |
| <b>5 Geometry-based Sorting Algorithm</b>                         | <b>29</b> |
| 5.1 Unlimited Communication Range . . . . .                       | 29        |
| 5.2 Limited Communication Range . . . . .                         | 32        |
| 5.3 Limited Communication Range with Range-based Safety . . . . . | 37        |
| 5.4 Experimental Results . . . . .                                | 40        |
| 5.4.1 Simulation . . . . .  | 40        |
| 5.4.2 Hardware . . . . .  | 42        |
| <b>6 Topology-based Sorting Algorithm</b>                         | <b>45</b> |

|          |   |           |
|----------|---|-----------|
| 6.1      | Algorithm . . . . .                           | 45        |
| 6.1.1    | Main Path Formation . . . . .                 | 46        |
| 6.1.2    | Geometric Operations (Robot motion) . . . . . | 49        |
| 6.1.3    | Main Path Topology Operations . . . . .       | 50        |
| 6.1.4    | Sub-tree Topology Operations . . . . .        | 51        |
| 6.1.5    | Movement of Endpoints . . . . .               | 54        |
| 6.2      | Analysis . . . . .                            | 55        |
| 6.2.1    | Safety . . . . .                              | 55        |
| 6.2.2    | Correctness . . . . .                         | 57        |
| 6.2.3    | Effectiveness . . . . .                       | 63        |
| 6.3      | Implementation . . . . .                      | 65        |
| 6.4      | Experimental Results . . . . .                | 66        |
| <b>7</b> | <b>Conclusion</b>                             | <b>70</b> |
|          | <b>Bibliography</b>                           | <b>71</b> |

# Illustrations

|     |  |    |
|-----|--|----|
| 1.1 | An example of physical sorting. Eight robots are initially randomly distributed in 2D space, and finally sorted to form a line between global minimum and global maximum (drawn in black). Line segments between robots represent communication network. . . . . | 2  |
| 3.1 | Position and heading. . . . .  | 8  |
| 3.2 | Bearing, orientation, and distance. . . . .  | 9  |
| 4.1 | Position and heading. . . . .  | 11 |
| 4.2 | 8 receivers' scopes overlap and partition the nearby area into 16 sectors.   | 12 |
| 4.3 | Measuring bearing and orientation with eight transmitters and eight receivers on each robot. . . . .   | 13 |
| 4.4 | T4 and T5 transmit the same message payload and checksum with different orientation bits. Both orientation bits are received by R1 and R2. . . . .   | 14 |
| 4.5 | A photo of bearing and orientation experiment. . . . .   | 16 |
| 4.6 | Bearing readings versus actual. . . . .  | 17 |
| 4.7 | Histogram of bearing error. . . . .  | 17 |
| 4.8 | Orientation readings versus actual. . . . .  | 18 |
| 4.9 | Histogram of orientation error. . . . .  | 18 |

|      |   |    |
|------|---|----|
| 4.10 | <b>a:</b> Each bit in a message is actually a series of 38kHz square wave. <b>b:</b> Changing voltage can change transmit power. <b>c:</b> Changing duty cycle can also change transmit power. . . . .  | 19 |
| 4.11 | Messages with different transmit power attenuate at different distances.  | 20 |
| 4.12 | Message with range bits. . . . .  | 21 |
| 4.13 | Message with descending range bits. . . . .   | 22 |
| 4.14 | Message with ascending range bits. . . . .  | 23 |
| 4.15 | Integrated circuit inside a receiver. . . . .   | 24 |
| 4.16 | Multiple range bits at the same power setting. . . . .  | 25 |
| 4.17 | Range experiment with range bits 8x faster in the back. . . . .   | 25 |
| 4.18 | Automatic range experiment. The host robot is on the top left with USB cable. . . . .   | 26 |
| 4.19 | A relatively good range estimation. . . . .   | 27 |
| 4.20 | Robot can make distance estimation based on range bits. . . . .   | 28 |
| 5.1  | Robot 2 moves to the midpoint of its predecessor (robot 1) and successor (robot 3), while robot 3 moves to the midpoint of its predecessor (robot 2) and successor (robot 4). . . . .   | 30 |
| 5.2  | Numerically solving ODE with 7 robots gives this solution. . . . .  | 31 |
| 5.3  | At least one robot moving at full speed until sorted. . . . .   | 32 |
| 5.4  | <b>a:</b> Robot 2 moves to the midpoint of robot 1 and robot 3. Link between robot 2 and robot 4 disconnects, causing robot 4 isolated. <b>b:</b> Robot 2 and robot 9 move in the opposite directions, leaving robot 5 isolated. <b>c:</b> Robot 3 and robot 5 are local minimum and maximum. They move in the opposite directions, leaving robot 4 behind, even though they are final predecessor and successor of robot 4. <b>c:</b> Robot 2 and robot 9 move in the opposite directions and away from robot 5. Since robot 5 is still connected to robot 10, it becomes a local minimum. | 36 |



|     |   |    |
|-----|---|----|
| 5.5 | <b>a:</b> There is no way to solve this problem, for there is no range overlap between robot 1 and robot 4, thus robot 2 and robot 3 cannot safely swap positions. <b>b:</b> It is too risky for robot 2 and robot 3 to make a swap, since unsafe ranges overlap but safe ranges do not overlap. They will stop at the boundary between safe range and unsafe range. <b>c:</b> This problem is solvable because of safe range overlapping. Safety feature prevents one normal robot leaving the safe range overlap before another robot enters it. . . . .                | 39 |
| 5.6 | Robot 2 and robot 3 are local minimum and maximum. They move to each other and cause disconnection. Range-based safety causes deadlock and cannot solve this problem. . . . .   | 39 |
| 5.7 | <b>a:</b> Convex hull of the robot configuration is shown in grey. Our distributed controller always reduces the area of the convex hull, which converges to zero if robots are in a line. <b>b:</b> Area of convex hull versus time for the same initial configuration with three different simulation mode: unsafe (limited range), safe (limited range with range-based safety), and unlimited range. . . . .  | 41 |
| 5.8 | Physical sorting with eight robots. Collision and interference happened, but robots managed to get sorted. . . . .  | 42 |
| 5.9 | <b>a:</b> Time spent versus number of robots. For each number of robots, 10 experiments with the same initial configuration are performed, and mean and standard deviation of time spent are shown in the figure. When number of robots increases, collision and interference become a severe problem and cause failure. <b>b:</b> Area of convex hull among time in one experiment. It decreases as robots become sorted, while physical interference may cause it to increase. 7% of the area cannot be removed due to sensor resolution and measurement error. . . . . | 43 |

|      |   |    |
|------|---|----|
| 6.1  | Topology-based sorting. . . . .   | 46 |
| 6.2  | The primary tree drawn in solid arrows and the feedback tree drawn in dashed arrows. Arrow from child to parent means child stores parent's label, while information flows in the opposite direction. The main path is highlighted as bold arrows. Neighbor connections not used to form trees are drawn with thin dashed lines. . . . .                    | 47 |
| 6.3  | Main path operations. . . . .   | 50 |
| 6.4  | Sub-tree operations. . . . .  | 52 |
| 6.5  | Sub-tree collapse. $m$ could be any robot on the main path (not necessarily $p$ 's parent), and $p$ could be any hops away from the main path. . . . .  | 53 |
| 6.6  | <b>a:</b> This configuration cannot be solved unless we shorten the main path to gain at least one more connection. <b>b:</b> Sortable after global maximum moved (degenerated triangles shown with curved edges). Detailed solving process will be shown in Fig.6.8. . . . .   | 54 |
| 6.7  | IDs of three robots forming a triangle have relative order, and we denoted 1, 2, and 3 as their ascending order. There are only six permutations of three different numbers and three different topologies for triangles adjacent to the main path, so we have eighteen local triangle states. There are no loops of operations between any states. . . . . | 58 |
| 6.8  | Robot 2 and robot 4 may perform operations sequentially or concurrently. All three cases will end in correctly sorted main path. . . . .  | 60 |
| 6.9  | Area of convex hull versus time. Each of ten configurations shown in different color is done with both algorithms. . . . .  | 68 |
| 6.10 | Topological measurements on the new algorithm. . . . .  | 69 |

## Tables

# Chapter 1

## Introduction

Arranging robots in a specific order is a necessary routine in some applications on multi-robot systems. For example, robots may need to be ordered to go through a narrow passage, or to perform sequential procedures. If the robots are homogeneous, then swapping tasks can solve this problem in some applications, but sorting is required if robots have different structures, are carrying different physical loads that cannot be swapped (and perhaps need to arrive in order), or differ in some other intrinsic quantities, such as remaining battery level that cannot be transferred via communication [1].

Sorting requires geometry measurement between neighboring robots. Local geometry measurement and communication is very useful on large swarm of low-cost robots, where global positioning is not available. Robots need to communicate with neighbors, as well as measure their relative positions. We need inexpensive and compact sensors to measure, in order to keep robots low-cost and scalable. Since real applications involve robots forming a network greater than individual's sensing range, and people study this effect in laboratories with confined space, we are focused on looking for sensors with short range. With sensor readings, robots can estimate relative positions of neighbors and build a map of local geometry.

Using only local geometry measurements, I describe a sorting algorithm to arrange robots into a line, evenly distributed, and sorted based on their labels. Fig.1.1 shows the initial state and final state of eight robots.

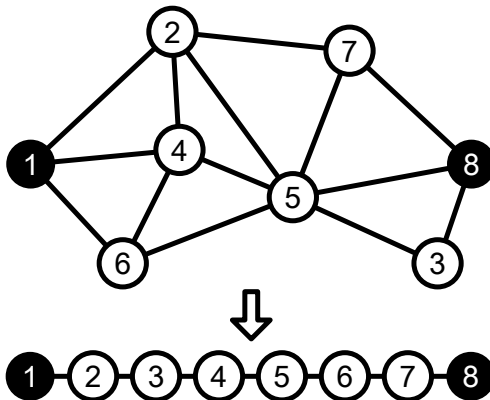


Figure 1.1 : An example of physical sorting. Eight robots are initially randomly distributed in 2D space, and finally sorted to form a line between global minimum and global maximum (drawn in black). Line segments between robots represent communication network.

Since I would like to deploy my algorithm on real robots, I am focused on a continuous model that can be adopted on wheel-based robots. They neither require a global synchronized clock nor depend on any global positioning. We require the initial configuration is a connected graph, and the algorithm preserves connectedness.

My contribution of this thesis are: range measurements on the  $r$ -one robot, geometry-based sorting algorithm, and topology-based sorting algorithm.

Because most previous work is based on unlimited communication range, which is impractical for large swarm of robots, I intended to develop a sorting algorithm for robots with limited communication range. I addressed on the issues caused by limited communication range, and came up with a straightforward geometry-based algorithm. This algorithm works well on real robots with general configurations, but it has theoretical defects that may cause disconnection in some special cases. This algorithm is also very difficult to prove convergence.

The topology-based algorithm takes advantage of explicitly building a main path,

in order to guide robots to form a line. I added topological operations to move robots around and get sorted. I demonstrate that this algorithm is robust and safe for any arbitrary initial robot configuration, without using mutual exclusion or consensus. I made minimal assumptions on the robot's communication and computation requirements, so my approach is suitable for simple robots with limited resources.

## Chapter 2

### Related Work

There are many previous works for robot measurements based on a variety of sensors. Major methods are about estimating the distance between robot and obstacle. Matijevics [2] and Sabatini [3] use infrared sensors with analog output signals. Those sensors provide very accurate distance measurement, but require additional analog circuit or analog-compatible microprocessors, and both the sensor and processing hardware are expensive. Some approaches require the robot or the sensor to rotate in order to scan the environment. Garcia [4] rotates the robot with a pair of infrared transmitter and receiver to estimate the distance to a planar obstacle. Novotny [5] used the Phong Illumination Model to draw a map of obstacles near the robot when robot rotates in place. Rotating the robot could overcome the limitation on a single sensor, but is time-consuming, requiring the robot to stop and rotate in place to get measurements, so it is not a good solution for multi-robot cooperation. Korba [6] developed a sensor with rotatable mirror. Vaz [7] put sensors on a rotatable chassis, and deploy multiple rotatable sensors on a robot to calculate distance based on triangle geometry. These kinds of sensors rotate much faster than robot moves, so that they can measure while robot is in slow motion. But their sensors are not only larger than our whole robot but also cost more than a single robot we use. In [8], pairwise sensors are used to for a single measurement, and difference between pairwise readings can be used to estimate distance and angles. Pairwise sensors are separated 15cm away, making this method unsuited for small robots. Using a sensor array [9] [10] [11] could

give accurate estimation and even take a photo under infrared wave band, but sensor array costs multiple times than a single sensor. Even they claim to be low-cost, a sensor array is too large to mount on a small robot.

There are also many works related to infrared wireless communication. Li et al. [12] provides an encoding method to communicate over infrared channels. Their message modulation protocol is similar to ours, but they are focused on generating the signal instead of receiving and processing, while my work is mainly about finding an optimized set of transmitter parameters to fit the receivers. Takai [13] uses directional sensors to reduce interference. Sensors are confined with barriers to reduce effective angle slice, and can rotate to face specific partner. Arvin [14] provides a method to communicate between robots and detect obstacles as well as to measure the distance, which is similar to our goal, but provided that infrared receivers can output analog signals.

There is limited previous work on sorting groups of robots. The most relevant work is of Litus and Vaughan [1], which uses a double bracket flow to build a dynamical system to model the robots' positions. Evolving this system drives the positions to a sorted ordering. This is a compact, analytical solution and has provable properties, but it requires that the robots are initially placed among a line parallel to a known axis. Additionally, it requires infinite sensing and communication range between robots, which makes it impractical.

McLurkin [15] presented the idea of physical sorting in his Ph.D. thesis. He implemented sorting on a group of expensive robots with accurate sensors, and most of his experiments succeeded due to relatively large communication range. However, he did not address on the potential failures, nor gave a proof or time complexity.

McLurkin, Li, and Embree modeled the system with a discrete model [16]. In this



model each robot jumps to its destination, which is the midpoint of predecessor and successor, at each time step. The authors proved convergence with matrix iteration, but also assumed a robot can communicate with its final predecessor and successor. These assumptions make it mathematically provable but impractical on real robots, since most ground robots are wheel-based or foot-based and can only move in some bounded space in a give time period.

To avoid disconnecting the network among the sorting algorithm, I am inspired by papers related to maintaining connectivity. Poduri [17] developed an algorithm to deploy a mobile sensor network with maximum area coverage, and robots are connected with any user-defined degree of connection. This could generate a connected uniform distribution when all robots stop. Zavlanos [18] and Williams [19] use potential field to keep robots connected and follow the leader, and maintain a similar organized structure in motion. Zavlanos [20] also keeps robots connected when flocking, which is a fully distributed algorithm without a leader. Hsieh [21] and Zhang [22] keep robots connected to each other while moving in an environment with obstacles. Robots maintain appropriate distances and relative angles to minimize the risk of disconnection caused by obstacles.

I am also inspired by formation control algorithms in order to straighten the sorted line. Chain formation algorithms provide multi-hop connection between two end-points. Kutylowski [23] provided a theoretical method to shorten the chain between an explorer and a base camp. Degener [24] introduced Move-On-Bisector strategy to straighten the chain, validated his algorithm keeping the chain connected, and proved time complexity. Dynia et al. [25] employed the Go-To-The-Middle strategy, when distance measurements are available as well as angle measurements. The chain can be tightened around obstacles and keep robots connected. They also proved that the

chain cannot break and always converges to minimal under their algorithms. [26] shows an algorithm to connect multiple endpoints with optimal set of chains. There are also other works about robot formation in other shapes. Balch [27] focused on unmanned ground vehicles to keep them in desired formation while moving in unknown environment. Balch also introduced social potential in his later work [28] to allow a group of robots to avoid obstacle with minimal deformation. Egerstedt [29] provided a mathematical model for multi-agent formation control, and applied to rigid body constrained motions. Ji [30] keeps robots connected during the rendezvous maneuver by adding appropriate weights to the connections.

To bring robots closer to each other, I referred to some work about multi-robot recovery. Ding [31] provided a preplanned recovery scheme to recover robots with multiple redundant trees. Cornejo [32] introduced  $k$ -redundancy to tolerate  $k$  robots failure. Habibi [33] presented an efficient way to recover all robots back to the base station with minimum disconnection. In my simulation a simple recovery is done by moving robots to their parents in a spanning tree, but employing those safe recovery methods could largely enhance safety on real robot platforms.

## Chapter 3

### Model and Assumptions

Global geometry is the configuration of robots in some give global coordinates. Each robot  $u$  has three degree of freedom in 2D Euclidean space: *position*  $(x_u, y_u)$ , and *heading*  $H_u$ , where heading denotes the front direction of the robot (Fig.3.1). A robot does not know its position or heading in any global coordinates, so it takes its front direction as  $x$ -axis in its own local coordinates.

Local geometry is the relative positions and directions between adjacent robots (Fig.3.2). *Bearing* is the relative direction of the remote robot in the local robot's local coordinates. *Orientation* is the relative direction of the local robot in the remote robot's local coordinates. Zero bearing and zero orientation means robots are facing to each other. *Distance* is measured between centers of two robots.

There are some relations between local geometry and global geometry, such as:

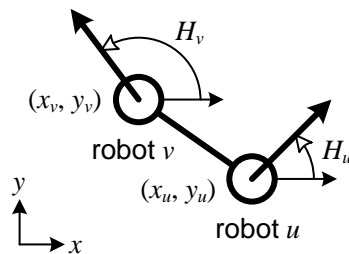


Figure 3.1 : Position and heading.

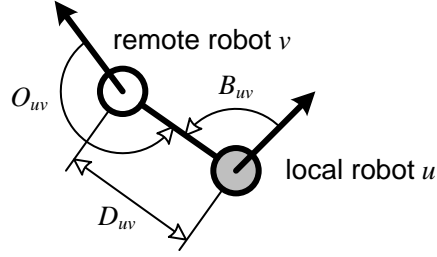


Figure 3.2 : Bearing, orientation, and distance.

$$B_{uv} = \arctan \frac{y_v - y_u}{x_v - x_u} - H_u \quad (3.1)$$

$$D_{uv} = \sqrt{(x_v - x_u)^2 + (y_v - y_u)^2} \quad (3.2)$$

$$H_u + B_{uv} = H_v + O_{uv} - \pi \quad (3.3)$$

Also between local coordinates, we have  $B_{uv} = O_{vu}$  and  $B_{vu} = O_{uv}$ .

There are  $n$  robots  $v_i, i \in \{1, \dots, n\}$ , with initial positions described by  $p_i = (x_i, y_i)$ . The communication network is an undirected graph  $G = (V, E)$ . Each robot is modeled as a vertex,  $u \in V$ , where  $V$  is the set of all robots and  $E$  is the set of all robot-to-robot communication links. The *neighbors* of each vertex  $u$  are the set of robots within line-of-sight communication range  $r_{\max}$  of robot  $u$ , denoted  $N(u) = \{v \in V \mid \{u, v\} \in E\}$ . Robot  $u$  sits at the origin of its local coordinate system, with its  $x$ -axis aligned with its current heading. Each robot is autonomous, using its own local geometry measurements to make decisions.

When I mention *unlimited communication range*, I have  $r_{\max} = +\infty$ , or  $r_{\max}$  is large enough to cover all other robots at any time. In this case, the communication

network is fully connected. When I mention *limited communication range*, the communication network does not have to be fully connected, but it is still path-connected, and bidirectional communication between robots exists if and only if their distance is smaller than  $r_{\max}$ . This assumption also results in monotonic communication graph, in which two connected robots continue to be connected if one of them moves directly towards the other.

I model the motion using a differential drive chassis, with translation speed up to  $v_{\max}$  and rotation in place. Robot has the ability to move toward one other robot, or move to the midpoint of two other robots.

Algorithm execution occurs in a series of synchronous *rounds*. This greatly simplifies analysis and is straightforward to implement in a physical system [15]. At the end of each round, every robot  $u$  broadcasts a message to all of its neighbors, and also receives a message from each neighbor  $v \in N(u)$ . Each message contains a set of public variables, including the sending robot's label  $ID(u)$ . The remaining variables will be defined later, but the total message has constant size.

To make my description and proofs simpler, I model each robot as point robot in my analysis and simulation, which allows me to ignore collision and communication obstruction. I assume that all the measurements and communications are accurate and timely without interference. I also ignore any time spent on rotating, and treat robots to be able to move at any direction. These assumptions do not hold for real physical systems; I will talk about these influences in the experiments.

## Chapter 4

### Local Geometry Measurement

#### 4.1 The r-one Robot

The r-one robot is a low-cost robot platform designed at Rice University (Fig.4.1) [34]. It has infrared for local geometry measurement and short-range communication, radio for long-range communication and remote programming, and odometers for feedback motion control. Actuators are recently designed for multi-robot manipulation [35].

The r-one robot uses infrared transmitters and receivers for short-range communication, since infrared sensors are much less expensive than camera or laser, are compact in size, and are much easier to process the data. On the r-one robot, infrared can provide bearing, orientation, and distance measurements to neighbors, as well as detecting obstacles. Since spaces in laboratories are usually small and con-



Figure 4.1 : Position and heading.

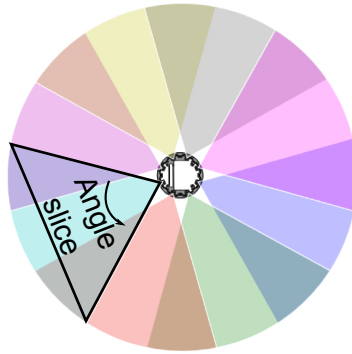


Figure 4.2 : 8 receivers' scopes overlap and partition the nearby area into 16 sectors.

finer, a practical concern is to make this measurement short-range with minimum interference.

Those infrared transmitters and receivers in TV remote controller seem to be a good solution. They are compact in size, easy to obtain at very low cost, and compatible with lithium-ion battery voltage. They are also designed to exchange messages in a short range. Theoretically, multiple sensors can be used to measure angles, and changing the transmit power could result in different sensing range. In this work where I developed a method to measure distance, I found that the electrical characteristics of the infrared receivers were interfering with the purpose of measurement. I did a lot of experiments to understand those effects and found optimal parameters to get better results.

There are eight IRL80A infrared transmitters from OSRAM GmbH [36] on the r-one robot, oriented around the circumference at 45 degrees. There are also eight GP1UX311QS infrared receivers from Sharp Electronics [37], each positioned between two transmitters. Each receiver can receive signal in a  $68^\circ$  arc, and multiple receivers overlap and partition the nearby area into 16 sectors (Fig.4.2). Therefore, bearing and orientation measurements are theoretically discrete, with a resolution of  $22.5^\circ$ .

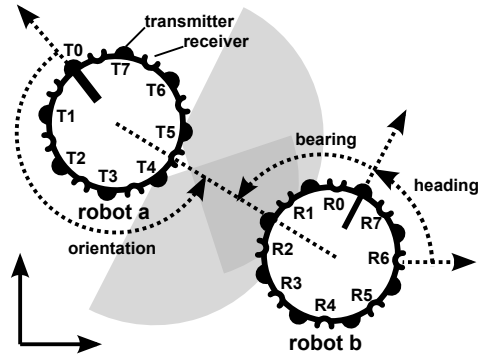


Figure 4.3 : Measuring bearing and orientation with eight transmitters and eight receivers on each robot.

Those transmitters and receivers are the same found in TV remote controllers, and they can be acquired easily at very low price.

Previously, there was no distance measurement on the r-one robot. Although most distributed algorithms work without distance measurement [38] [39] [40] [41] [42], having distance measurement could highly improve precision and efficiency [43]. Since infrared messages with lower power settings attenuate at shorter range, if one robot sends messages with different power settings, the other robot could take the discrete distance measurement based on which messages are received. However, because of the complexity of electrical characteristics in the infrared receivers, and messages are sent with random offsets, reaching an optimal set of message design is very difficult.

## 4.2 Bearing and Orientation Measurement

To implement bearing and orientation measurement, we need to know which receivers can receive messages from a neighbor, as well as which transmitters transmitted those messages (Fig.4.3). In this example, robot *b* receives messages with R1 and R2, so it takes the average direction of R1 and R2 as the bearing of robot *b*. Robot *b* also



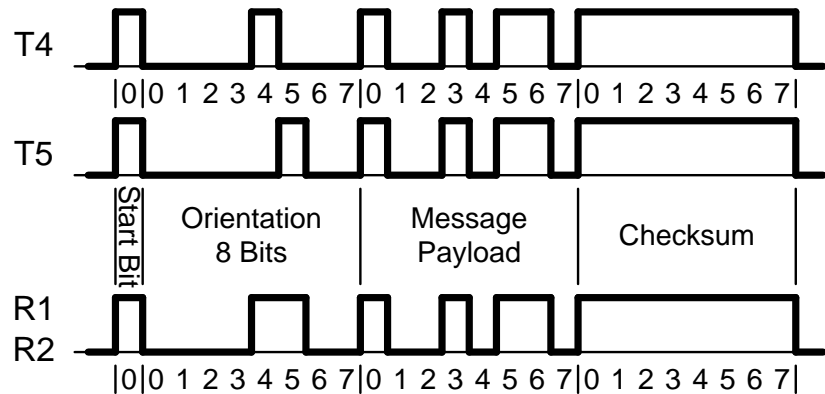


Figure 4.4 : T4 and T5 transmit the same message payload and checksum with different orientation bits. Both orientation bits are received by R1 and R2.

knows the orientation of robot  $a$ , if messages from T4 and T5 are different and each claims which transmitter the message is transmitted from.

The transmitting program transmits a message in the following time schedule: First, all the transmitters turn on for 800us, as the *start bit*. Second, T0 to T7 alternatively turns on for 800us, as eight *orientation bits*. Third, *message payload* is transmitted on all transmitters, usually the sender's ID. Fourth, a *checksum* (CRC [44] or other algorithm) of the message payload. The transmitting program pauses for some random time period (average about 200ms) before sending another message, in order to reduce medium conflict.

The receiving program is triggered if any receiver begins to receive infrared beam, normally the start bit. Then it runs at every 800us, gathers all the states for each receiver, and writes them into the receiving buffer. Fig.4.4 shows a message received by R1 and R2, while orientation bits indicate the message is from T4 and T5.

*Receiving matrix* is defined as an adjacency matrix between transmitters on the remote robot and receivers on the local robot for a transferred message. It can be retrieved from the receiving buffer. One item at position  $(i, j)$  is marked 1 if and

only if receiver  $i$  received the message with orientation bit  $j$  (from transmitter  $j$ ). *Receiving bits* are defined as a vector of which receiver reports received a message.

Normally, checksum should match with message payload, and receiving bits should match with receiving matrix. However, in practice, any bit could be flipped due to medium conflict or sample mistake. If the checksum does not match with message payload on one receiver, the robot tries to recover the message with information from other receivers, and discards the message if it cannot be recovered. If the orientation bits are missing, orientation cannot be calculated. In this case, receiving program reports that orientation is unknown. If robot extracts bearing from receiving matrix, bearing information may also be lost, so the robot always uses receiving bits to calculate bearing, and receiving bits are always available. (Bearing may be unavailable if directions of receiving bits sum up to zero.)

In addition, reflection may cause messages delivered in multiple paths. If we have receiver bits or orientation bits with 3 bits, two are adjacent and one is separate, the program can remove the obvious error bit. Rapid changes beyond the angle speed that robot can rotate are also filtered out. But in other combinations, the program cannot distinguish the error bit. It either returns unknown bearing/orientation or returns its best guess.

In order to know the relationship between readings and actual bearing/orientation, I performed experiments to get both readings and actual data. Two robots are put on two mechanical turntables (Fig.4.5), and their distance can be manually controlled. Each turntable can be turned on, and it rotates slowly at constant speed. The receiving robot is plugged into a computer to get access to data log. If we rotate the receiving robot, we can plot bearing readings versus actual. If we rotate the transmitting robot, we can plot orientation readings versus actual. I also record the time that

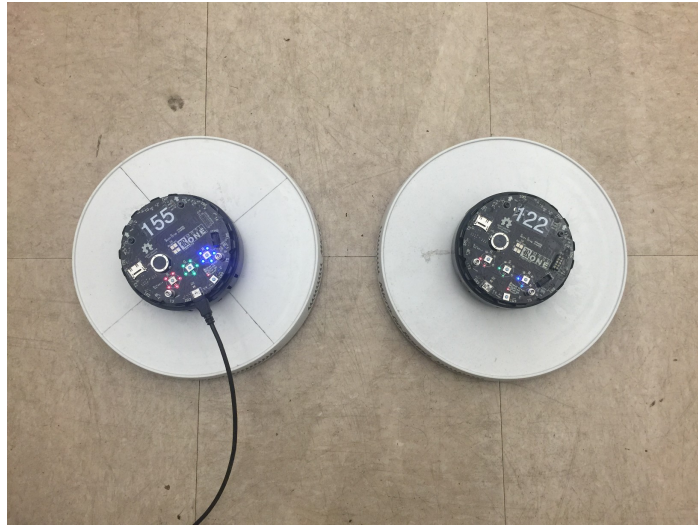


Figure 4.5 : A photo of bearing and orientation experiment.

the turntable rotates for a full circle, so that I can match the readings with actual angles.

The result of bearing is shown in Fig.4.6. In the plot of bearing readings versus actual, most data have one or two active receivers, and they form the major strips. Some data have three or more receivers and the averages are away from the major strips.

We calculate the error of each data point, and classify them into a histogram (Fig.4.7). The standard deviation is about  $14.3^\circ$ .

The result of orientation is shown in Fig.4.8. In the plot of orientation readings versus actual, most data have one or two active transmitters, and they form the major strips. Some data have three or more transmitters and the averages are away from the major strips. Data with unknown orientation are not included.

We calculate the error of each data point, and classify them into a histogram (Fig.4.9). The standard deviation is about  $13.9^\circ$ .

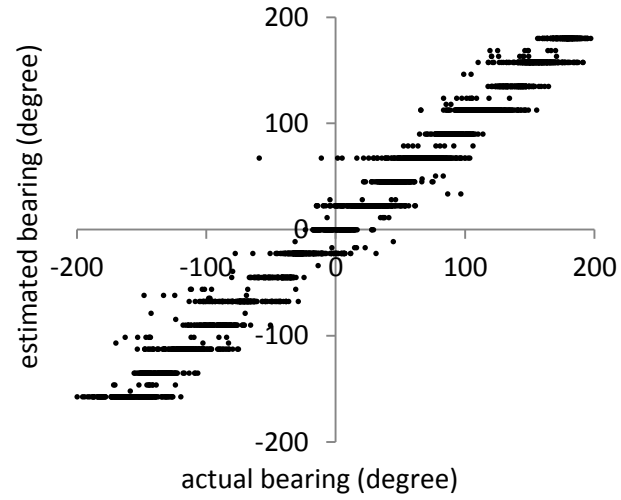


Figure 4.6 : Bearing readings versus actual.

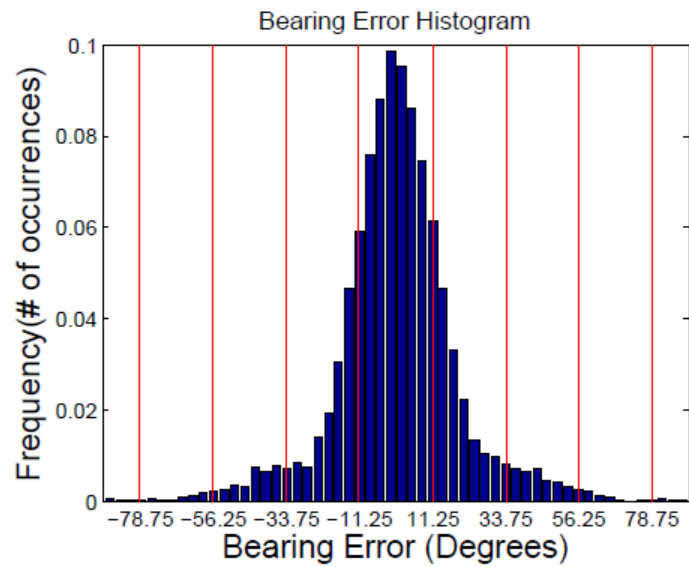


Figure 4.7 : Histogram of bearing error.

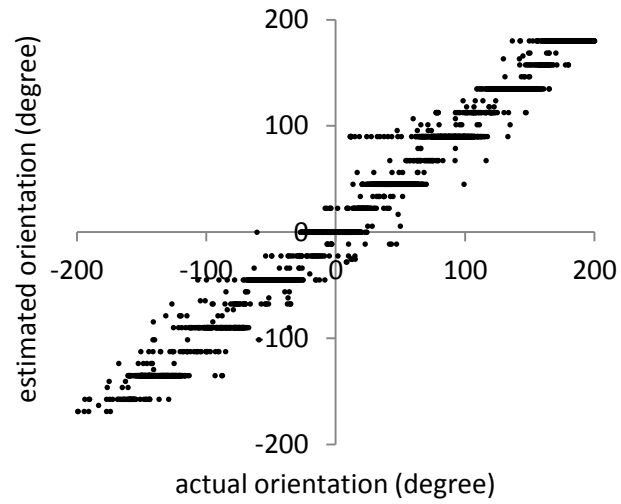


Figure 4.8 : Orientation readings versus actual.

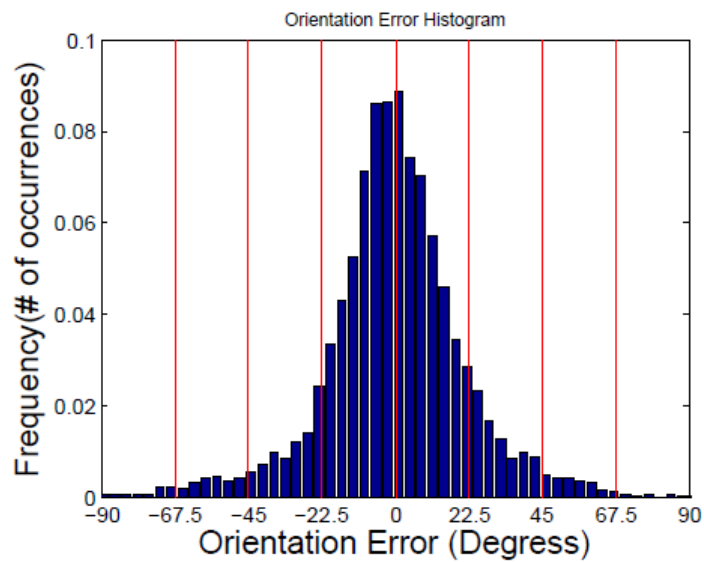


Figure 4.9 : Histogram of orientation error.

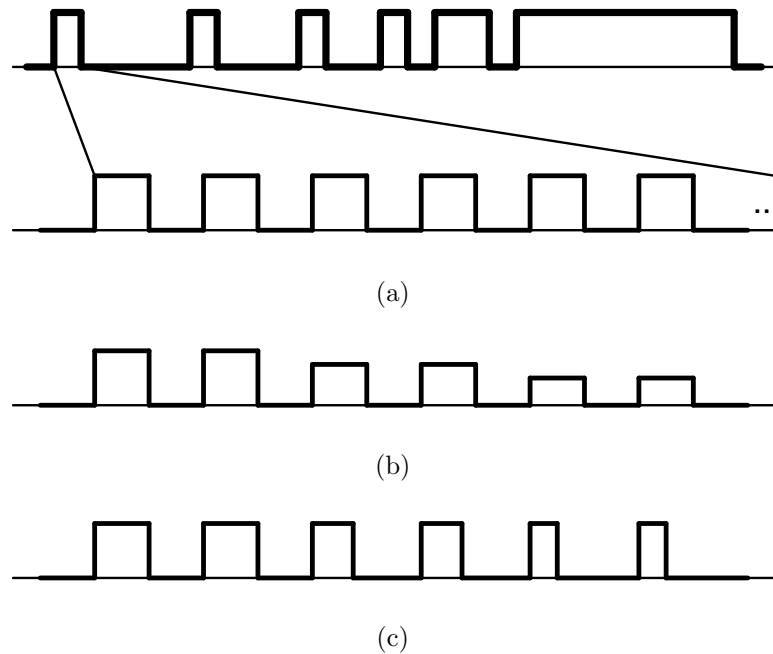


Figure 4.10 : **a:** Each bit in a message is actually a series of 38kHz square wave. **b:** Changing voltage can change transmit power. **c:** Changing duty cycle can also change transmit power.

### 4.3 Distance Measurement

Transmitters transmit messages modulated on a 38kHz square wave (Fig.4.10), while receivers only receive signals at 38kHz to avoid interference from heat sources. Changing transmitters' voltage could result in different transmitting range, but it requires analog circuit to change voltage, and is difficult to implement on hardware. Since I have control over the Pulse-Width Modulation (PWM) hardware, I can change the duty cycle of the square wave, in order to change the transmit power. (Receiver sensitivity cannot be adjusted by changing supply voltage.)

First I tried to send different messages with different transmit power. In each trail I set the power setting for the whole message, put the robots at different distances, and record the percentage of messages that can be received by the receiving robot

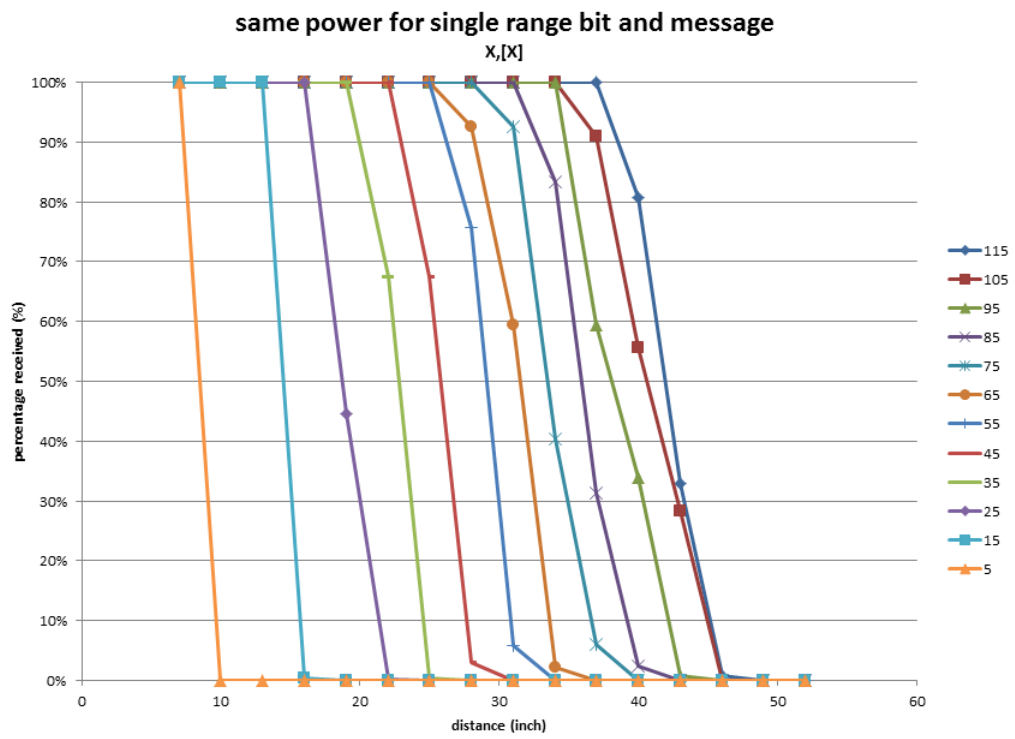


Figure 4.11 : Messages with different transmit power attenuate at different distances.

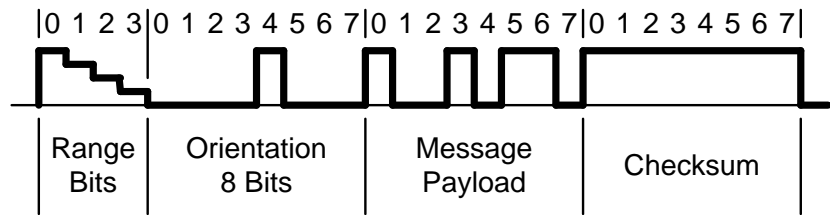


Figure 4.12 : Message with range bits.

(provided the transmitting robot always sends four messages in one second). Twelve trails with different transmit power are shown in Fig.4.11. (Duty cycle is the power setting divided by 512.) Each trail attenuates sharply, and lower transmit power results in shorter range.

If robots change transmit power for each message, the cost is too high. It not only takes many messages to measure the distance, but also robots need to encode power setting value inside the message. Bandwidth is very limited in infrared communication, and robots need long waiting time between transmitting to reduce conflict, so it is better to have as fewer additional bits as possible.

My solution is to replace the single start bit with multiple bits with different power settings, and I call them range bits (Fig.4.12). Bits with different transmit power are implemented as changing duty cycle of the 38kHz square wave instead of changing voltage. Number of range bits and their power settings can be manually set and should be shared between transmitting and receiving robots.

At first I tried to have 14 range bits with descending power settings, followed by a full power bit (15 total range bits), but there seems to be some abnormal peak at 85cm (Fig.4.13).

Then I tried making those range bits ascending with a full power bit in the front. The result is really surprising (Fig.4.14).



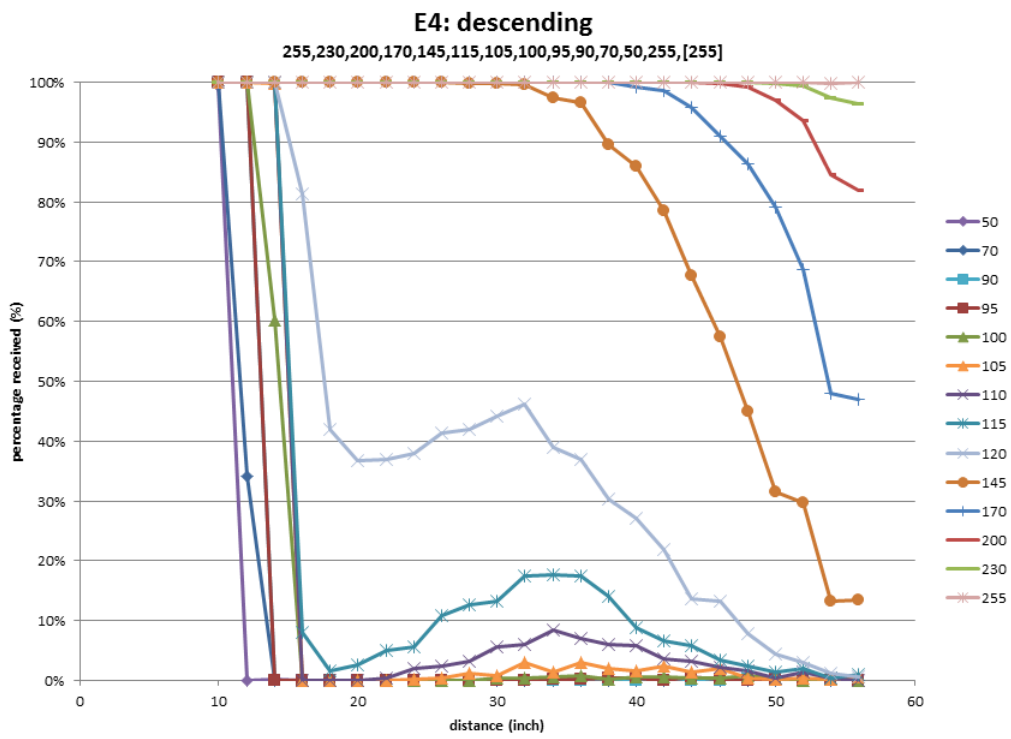


Figure 4.13 : Message with descending range bits.

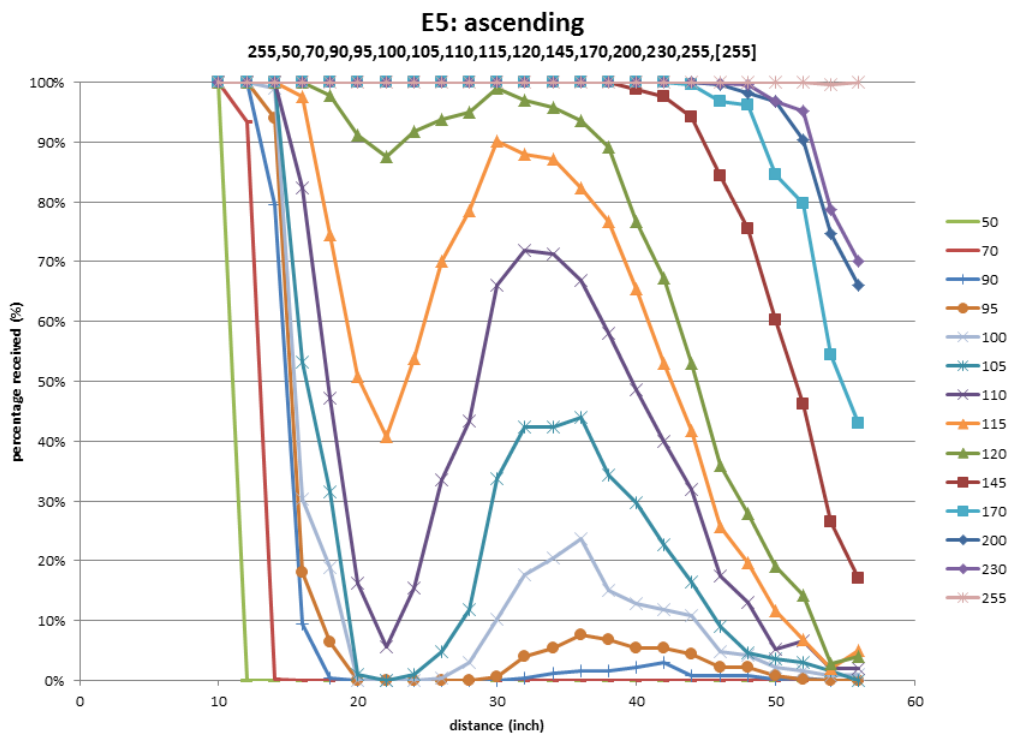


Figure 4.14 : Message with ascending range bits.

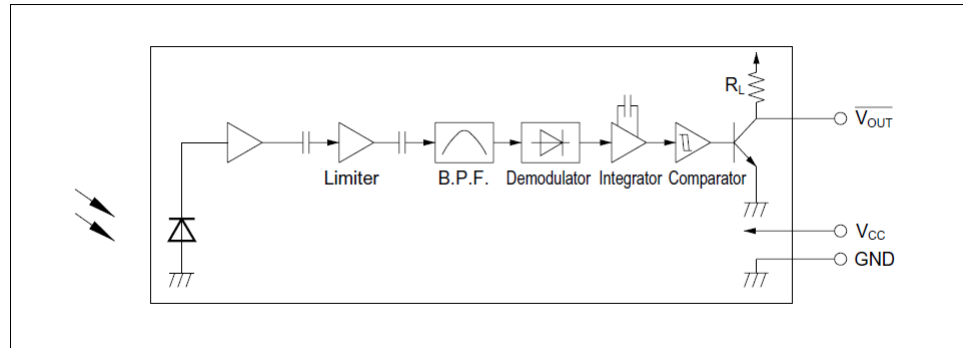


Figure 4.15 : Integrated circuit inside a receiver.

I previously assumed it could be some lens effect on the transmitter, but soon found this assumption contradicting with the first experiment in Fig.4.11.

One possible reason is that the receiver is not a single photodiode, but rather a complicated integrated circuit (Fig.4.15) as shown in its datasheet. It has an integrator to provide dynamic threshold. If we send a long series of 1's, the threshold gradually raises and sometime later the output becomes zero. The threshold gets lower if it does not receive anything in some time period. This is designed for infrared message encoding [45] [46], but not a good feature to use infrared receiver for range measurement. However, all similar low-cost products come with this feature.

So I did an experiment with multiple range bits with the same power setting. Indeed, latter range bits decay faster than former ones (Fig.4.16).

In order to beat the electrical characteristics of the receiver, one solution is to send range bits faster. Each range bit takes 100us, just 1/8 of other bits. Another solution is to put range bits in the back. This could lessen the dynamic change of threshold, but time jitter reduces accuracy. Fig.4.17 shows an experiment with range bits 8x faster and located in the back.

I developed a method to optimize range parameters automatically 4.18. Several

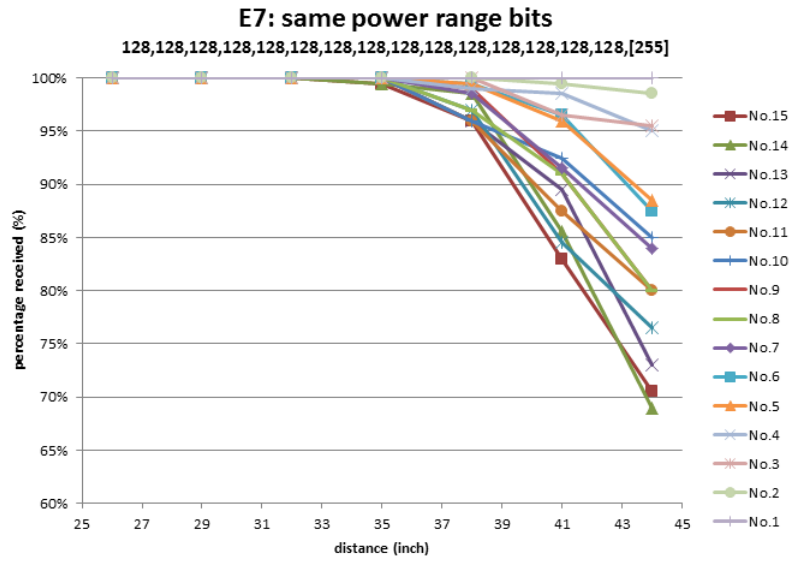


Figure 4.16 : Multiple range bits at the same power setting.

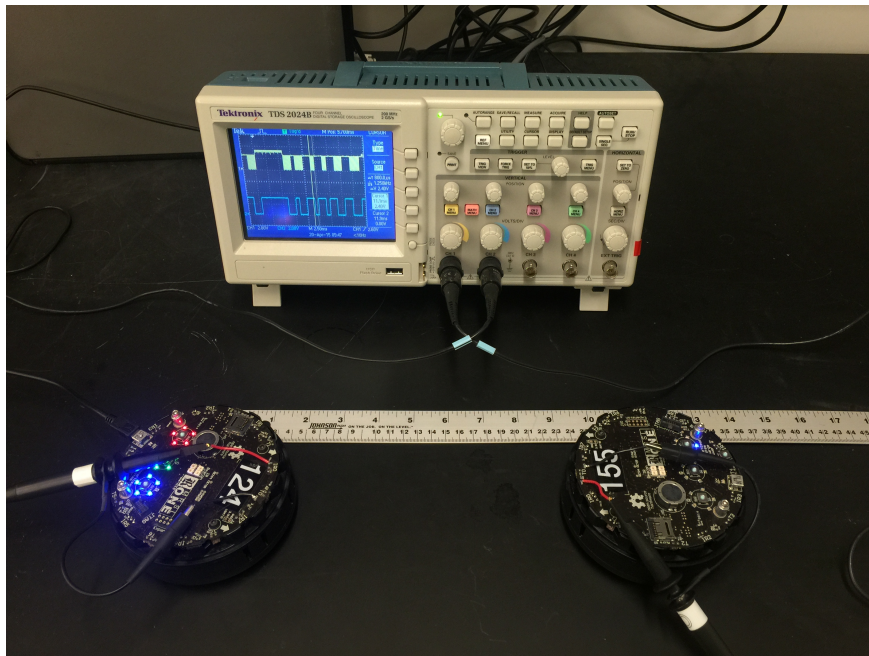


Figure 4.17 : Range experiment with range bits 8x faster in the back.

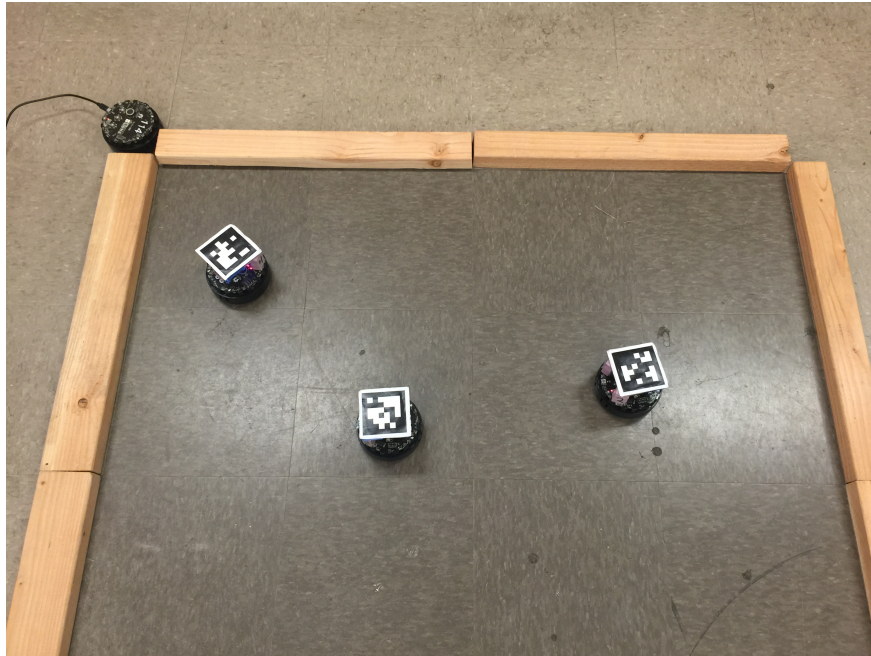


Figure 4.18 : Automatic range experiment. The host robot is on the top left with USB cable.

robots are placed on the ground with barrier, and the barrier is lower than the transmitters and receivers so that it does not reflect infrared beam back. When a robot contact with the barrier with bumpers, it rotates to random direction and then moves straight. Once a robot receives a message from another robot via infrared communication, it transfers local ID, remote ID, receiving matrix, receiving bits, and range bits to the host robot via radio communication. The host robot has its infrared system shut down, listens to radio communication, and relays the radio messages to the computer via USB cable. At the same time, an overhead camera takes pictures, and robots' positions are calculated with AprilTag positioning system [47]. Data from host robot and data from AprilTag are combined to get distance readings versus actual. The host robot can also remotely program other robots with a new set of parameters.

Fig.4.19 shows average and standard deviation of range estimation for publication

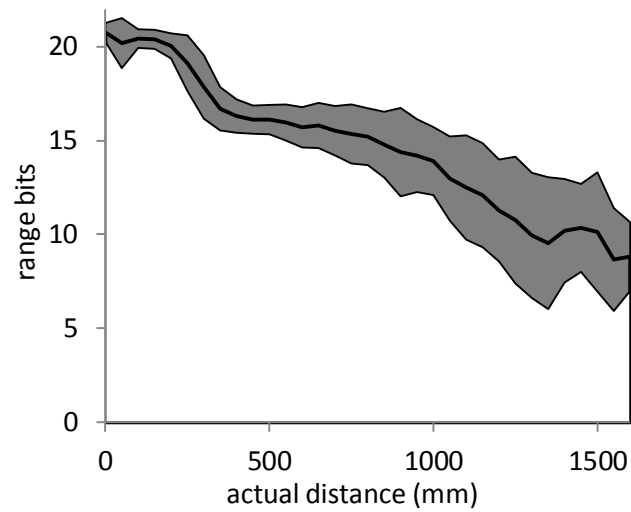


Figure 4.19 : A relatively good range estimation.

[35].

If we reverse Fig.4.19, we get Fig.4.20. This provides a method for robots to estimate distance based on range bits. Although standard deviation is relatively large, robots can still tell whether they are too close (less than 400mm), just right (400mm to 800mm), too far (800mm to 1200mm), or the connection is unstable (more than 1200mm). This result could improve precision on many distributed algorithms including my physical sorting algorithm.

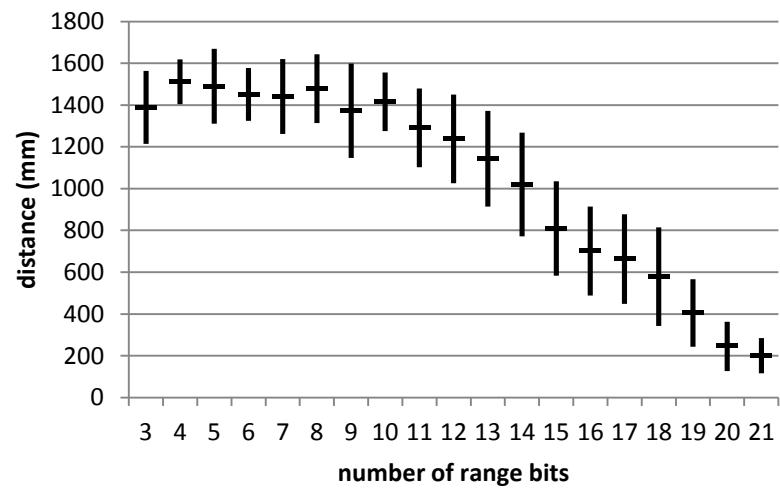


Figure 4.20 : Robot can make distance estimation based on range bits.

## Chapter 5

### Geometry-based Sorting Algorithm

With the ability to measure local geometry, a robot can measure the relative direction of the midpoint between two other robots. I introduce a straightforward sorting algorithm, and show it works in general cases but fails in special cases.

#### 5.1 Unlimited Communication Range

There are  $n$  robots forming a connected communication network. Each robot has some unique label,  $ID(i)$ . The total set of these labels is unknown, but there is a total order among them, *i.e.*, there is a permutation  $\pi : \{v_1, \dots, v_n\} \rightarrow \{1, \dots, n\}$ , such that  $ID(\pi(1)) < ID(\pi(2)) < \dots < ID(\pi(n))$ . The labels are the properties to get sorted, and I assume that robots have unique serial numbers to compare when their properties have the same value. To make my description simpler, I denote them as  $1, \dots, n$ , but remember that they do not have to be sequential. When I compare two robots  $u < v$ , I mean comparing their labels  $ID(u) < ID(v)$ .

Now the overall task is to achieve a sorted, evenly spaced arrangement of robots between those with lowest and highest labels, *i.e.*, robot  $i$  must move to position  $p_1 + (i - 1) \times (p_n - p_1) / (n - 1)$ , for  $i = 1, \dots, n$ .

First I consider a simple version of the sorting problem, in which robots have unlimited communication range. I define two robots, the *global minimum* and the *global maximum* are the two robots with the smallest and largest labels, respectively.



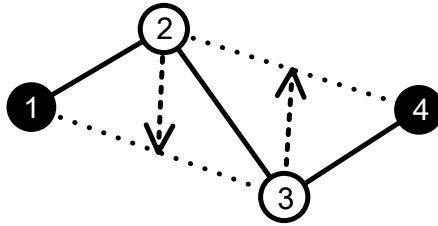


Figure 5.1 : Robot 2 moves to the midpoint of its predecessor (robot 1) and successor (robot 3), while robot 3 moves to the midpoint of its predecessor (robot 2) and successor (robot 4).

They remain in their original position to let other robots form a line between them. Each other robot (called *normal* robot) finds the robot with largest label below its own, and the robot with smallest label above its own. The robot treats these as *predecessor* and *successor*, respectively. If there is only one robot, it cannot find any other robots and become *isolated*. An isolated robot is trivially sorted and I will not consider this in my future discussion.

Li, McLurkin, and Embree [16] developed a jumping model where robots jump to the midpoint between their predecessor and successor in each synchronized round. They proved convergence with matrix iteration, while it may take infinite time to get converged. Inspired by their idea, I developed the continuous model and discrete motion model with similar method (Fig.5.1).

Each normal robot moves toward the midpoint between its predecessor and successor. This algorithm terminates when all normal robots are located within some radius  $\epsilon$  from the midpoint between their predecessors and successors, and they are sorted between global minimum and maximum.

The continuous model in 1-dimensional Euclidean space can be described with the following Ordinary Differential Equations (ODE), where  $v_x$  is some constant for speed scaling:

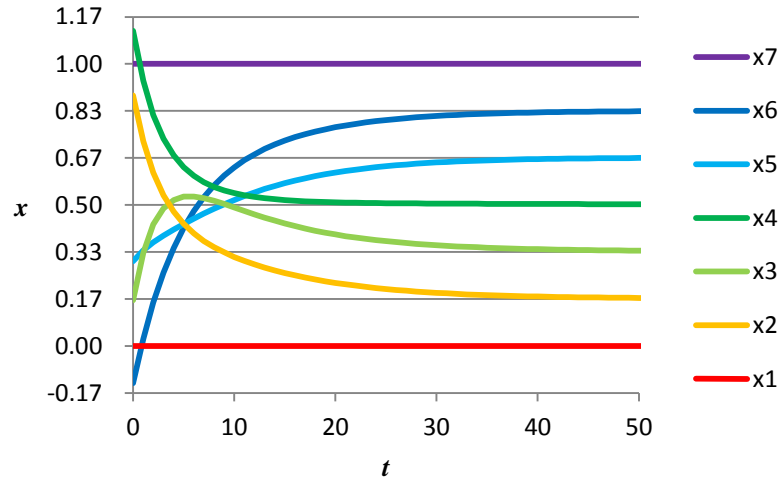


Figure 5.2 : Numerically solving ODE with 7 robots gives this solution.

$$\begin{aligned}
 x_1 &= 0, \\
 \dot{x}_k &= v_x \left( \frac{x_{k-1} + x_{k+1}}{2} - x_k \right), 1 < k < n, \\
 x_n &= 1.
 \end{aligned} \tag{5.1}$$

Solving those equations results in converging to sorted order. A numerical solution with 7 robots is shown in Fig.5.2. But if I model the system as ODE, robot moves at the speed proportional to the distance between itself and the midpoint, which is unbounded and is impractical.

If I set an upper bound on robot speed, and also require at least one robot moving at full speed unless already sorted, the numerical solution is shown in Fig.5.3. Robot moves to midpoint at full speed, and once at the midpoint, it moves at the speed the midpoint moves. This model is difficult to be solved with differential equa-

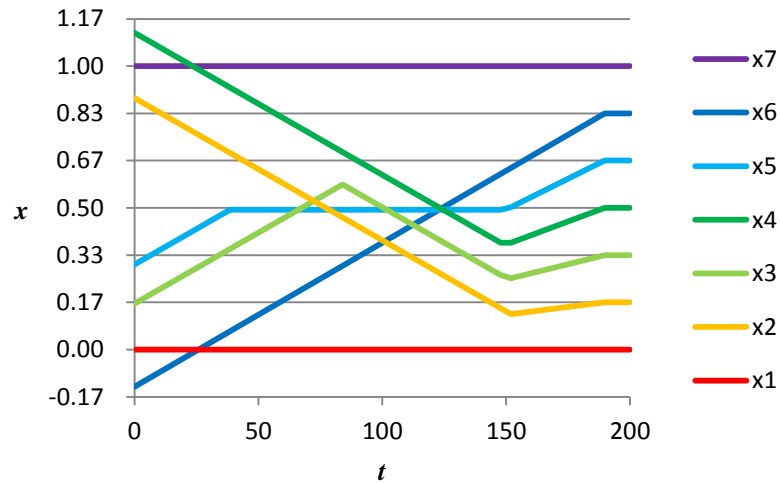


Figure 5.3 : At least one robot moving at full speed until sorted.

tions, especially in spaces with higher dimensions, where speed in each dimension is correlated.

In a stepping model, this procedure in each round is shown in Algorithm 1. Each robot falls in one of three states: normal, global minimum, and global maximum, if not trivially isolated. I assume that  $i$  is the ID of current robot,  $p_i$  is the predecessor, and  $s_i$  is the successor. I will use stepping model for all future discussions, since it is more closer to real robots.

## 5.2 Limited Communication Range

When robots have infinite sensing and communication range, or the range is large enough to cover all robots, a robot will immediately find whether it is the global minimum or maximum. Each other robot will immediately find its predecessor and successor. The final sorted network is immediately obvious, and failure from lost connections is not possible. However, only the smallest populations can have global

---

**Algorithm 1** LOCAL-STATE( $p_i, s_i$ )

---

```

1: if ( $p_i \neq \emptyset$ ) and ( $s_i \neq \emptyset$ ) then
2:    $state = \text{NORMAL}$ 
3:    $action = \text{MOVETOMIDPOINT}(p_i, s_i)$ 
4: else if ( $p_i = \emptyset$ ) and ( $s_i \neq \emptyset$ ) then
5:    $state = \text{GLOBAL MINIMUM}$ 
6:    $action = \text{STOP}$ 
7: else if ( $p_i \neq \emptyset$ ) and ( $s_i = \emptyset$ ) then
8:    $state = \text{GLOBAL MAXIMUM}$ 
9:    $action = \text{STOP}$ 
10: else
11:    $state = \text{ISOLATED}$ 
12:    $action = \text{STOP}$ 
13: end if

```

---

communications. Most multi-robot systems must use multi-hop communications to cover a large geographic area. The small sensing and communication range makes this problem more difficult.

When communication range is limited, three major issues occur. The first issue is that *final* predecessor (or successor) may not be initially in the communication range of a robot. We define final as the globally best fit and should be the neighbors when sorting finishes. The second issue is that no choice of predecessor (or successor) exists in the communication range of a normal robot, and it becomes a *local minimum* (or *local maximum*). The third issue is that robots may fall out of communication range, from the initial configuration or during the sorting algorithm. This causes disconnection and renders failure in sorting.

For the first issue, a normal robot simply picks *current* predecessor and successor of the best fit in its communication range. Once a better fit comes into communication range, the robot changes predecessor or successor to the better fit.

We address the second issue by running a distributed leader election algorithm

to discover the global minimum and maximum [15]. At the first round, each robot assumes it is both the global minimum and maximum, since it has no information about its neighbors. It broadcasts its claim with its label. At other rounds, a robot will receive messages from its neighbors. It compares those sources and may find a new minimum (or maximum). A robot stores and broadcasts its best knowledge of the global minimum (and maximum), its parent toward the global minimum (and maximum), the hops to the root, as long as its own label. If a robot is the minimum (or maximum) in its direct neighbors, but has learned that there is another robot with smaller (or larger) label, it marks itself as local minimum (or maximum). Each robot broadcasts its newest knowledge in each round.

After at most  $n$  rounds, all robots will discover the existence of global minimum and maximum, and the two robots with smallest label and largest label still consider themselves as global minimum and maximum in a connected network. This procedure also generates a spanning tree rooted at the global minimum (called the *min-tree*), and another spanning tree rooted at the global maximum (called the *max-tree*).

Navigation is introduced to solve the problem. When a robot becomes a local minimum (or maximum), there is another global minimum (or maximum) beyond its sensor range. A local minimum (or maximum) has different behavior: it moves toward its parent in min-tree (or max-tree) [48]. Since it will get closer to the global minimum (or maximum), it will find a new parent with fewer hops, or finally finds a predecessor (or successor) and becomes normal. It is guaranteed to find a predecessor (or successor) while repeating this procedure, since the global minimum (or maximum) is the last choice.

With limited communication range, each robot falls in one of five states: normal, global minimum, global maximum, local minimum, and local maximum, if not

---

**Algorithm 2** LOCAL-STATE( $p_i, s_i, minID, maxID$ )

---

```

1: if ( $p_i \neq \emptyset$ ) and ( $s_i \neq \emptyset$ ) then
2:    $state = \text{NORMAL}$ 
3:    $action = \text{MOVETOMIDPOINT}(p_i, s_i)$ 
4: else if ( $p_i = \emptyset$ ) and ( $s_i \neq \emptyset$ ) and ( $minID = i$ ) then
5:    $state = \text{GLOBAL MINIMUM}$ 
6:    $action = \text{STOP}$ 
7: else if ( $p_i \neq \emptyset$ ) and ( $s_i = \emptyset$ ) and ( $maxID = i$ ) then
8:    $state = \text{GLOBAL MAXIMUM}$ 
9:    $action = \text{STOP}$ 
10: else if ( $p_i = \emptyset$ ) and ( $s_i \neq \emptyset$ ) and ( $minID \neq i$ ) then
11:    $state = \text{LOCAL MINIMUM}$ 
12:    $action = \text{MOVETOPARENT}(min-tree)$ 
13: else if ( $p_i \neq \emptyset$ ) and ( $s_i = \emptyset$ ) and ( $maxID \neq i$ ) then
14:    $state = \text{LOCAL MAXIMUM}$ 
15:    $action = \text{MOVETOPARENT}(max-tree)$ 
16: else
17:    $state = \text{ISOLATED}$ 
18:    $action = \text{STOP}$ 
19: end if

```

---

isolated. We firstly run a spanning tree algorithm as in [49], and each robot learns  $minID$  and  $maxID$  as the global minimum and maximum labels from communication network. Then we use Algorithm 2 to implement navigation.

The third issue is the most difficult one to solve. Moving robots may cause network to disconnect. Fig.5.4(a) displays a simple example with only four robots, including a local maximum, will cause a disconnection. Even without local minimum or maximum, some configurations can still cause disconnection, such as in Fig.5.4(b), robot 2 and robot 9 move in the opposite directions in order to move to their midpoints. Robot 5 can catch up with the midpoint between robot 2 and robot 9, however, if both neighbors move out of range simultaneously, robot 7 is left isolated. Even a robot with final predecessor and successor in range may get lost (Fig.5.4(c)), due to its predecessor and successor are local minimum and maximum.

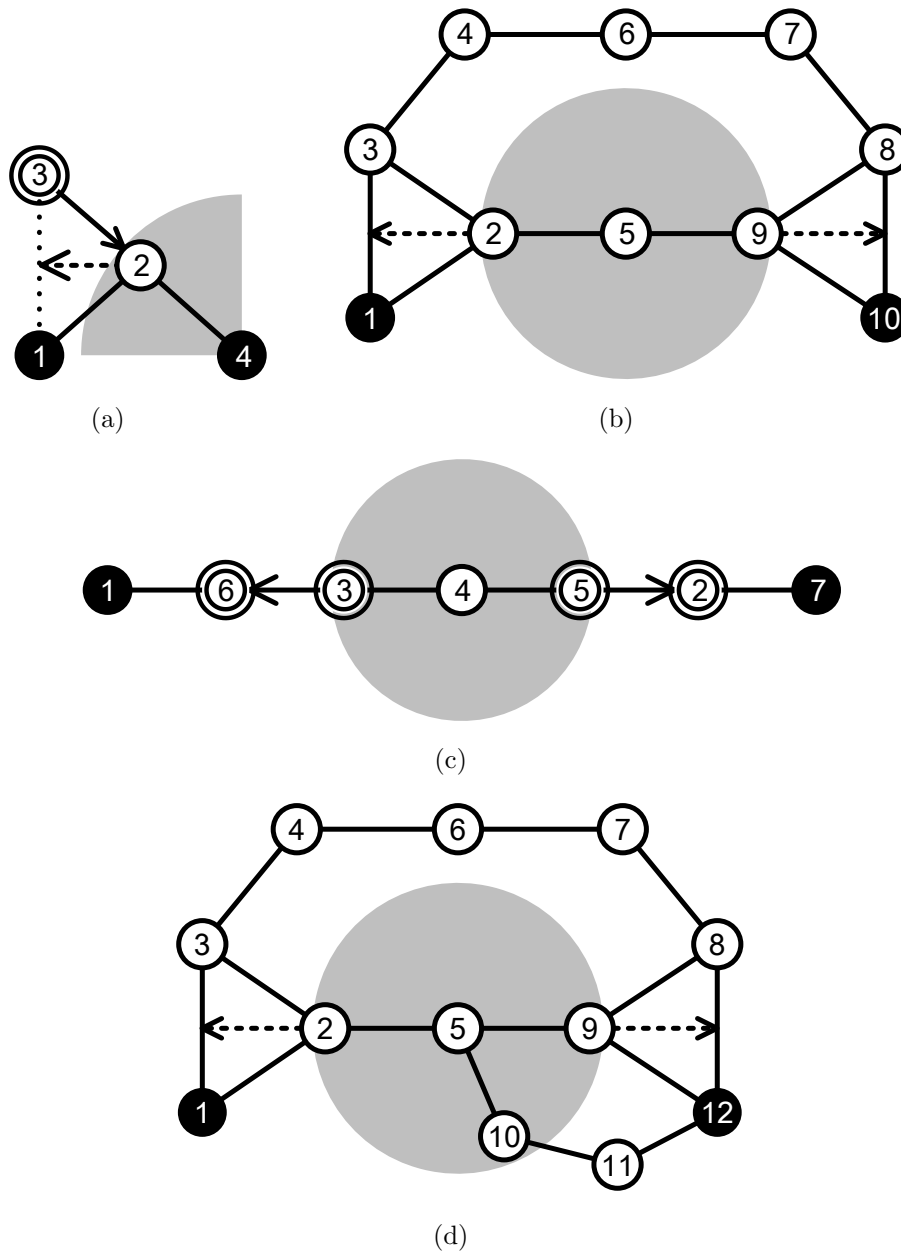


Figure 5.4 : **a**: Robot 2 moves to the midpoint of robot 1 and robot 3. Link between robot 2 and robot 4 disconnects, causing robot 4 isolated. **b**: Robot 2 and robot 9 move in the opposite directions, leaving robot 5 isolated. **c**: Robot 3 and robot 5 are local minimum and maximum. They move in the opposite directions, leaving robot 4 behind, even though they are final predecessor and successor of robot 4. **c**: Robot 2 and robot 9 move in the opposite directions and away from robot 5. Since robot 5 is still connected to robot 10, it becomes a local minimum.

It is also possible to generate local minimum or maximum during the procedure even if no local minimum or maximum exists initially (Fig.5.4(d)).

Thus, only if pairwise normal robots consider each other as predecessor/successor, their connection will never lose as long as this condition satisfies. If all robots have found their final predecessors and successors (except global minimum and maximum), a sorted path appears from the global minimum, visiting all robots in ascending order along the communication network, to the global maximum. Once sorted path exists, sorting problem is guaranteed to be solved, since no movement could tear apart pairwise predecessor/successor. But this assumption is as strong as unlimited communication range, so it is impractical in general.

### 5.3 Limited Communication Range with Range-based Safety

To maintain connectivity, robots may need to stop in some circumstances in order not to lose connection to a neighbor. A local decision process works to maintain connectivity in many circumstances. When a robot moves, it calculates the new distances to its neighbors under the influence of such movement. If the distance to a neighbor increases and is approaching sensor range limit, the neighbor is marked as risky. We place a threshold, called *safe-radius*, as a portion of sensor radius to decide whether connection is risky. It should be selected such that even if two robots with safe connection decide to move away from each other at the same round, they will not disconnect at the following round. This could eliminate the consensus problem. All other neighbors without such risk of disconnection are marked as safe.

Then for each safe robot, see whether that robot can connect to a risky neighbor, that is, estimate their positions and see whether their distance is smaller than sensor range. If they can be connected, change the risky neighbor to safe, and expand safe



neighbors from that robot. After this process, if there are any neighbors still marked risky, that robot cannot be safely reached from other neighbors. Such movement potentially disconnects the network and should be prohibited. The robot stops if this happens, and waits until an alternative path appears to allow it to move.

---

**Algorithm 3** CONNECTIVITY(*neighbors, safe-radius*)

---

```

1: for each robot  $r$  in neighbors do
2:   Estimate  $r.newdistance$  after next local movement
3:   if ( $r.newdistance > r.distance$ ) and ( $r.newdistance > safe-radius$ ) then
4:     mark edge( $i, r$ ) unsafe
5:   else
6:     mark edge( $i, r$ ) safe
7:   end if
8: end for
9: broadcast safe edges to neighbors and receive safe edges from neighbors
10: build spanning tree only with safe edges
11: if spanning tree fails to reach all neighbors then
12:   speed  $\leftarrow 0$ 
13: end if

```

---

If bearing measurement and distance measurement are too noisy, each robot can share their local neighbor list to direct neighbors, and mark each edge safe or unsafe. Thus, robots get exact local communication network instead of estimating inter-neighbor connections with bearing and distance, at a cost of communication  $O(|N(u)|)$  on each robot at each round. Distance measurement is only used for estimating whether a link is at risk. Algorithm 3 implements this method.

This safety method may cause deadlock when the configuration is so risky that it is nearly impossible to solve (see Fig.5.5(b)). However, I still find some example (see Fig.5.6) that may be solved by some off-line method, but cannot be solved with my safety constrain. This shows the limitation of geometry-based sorting algorithm.

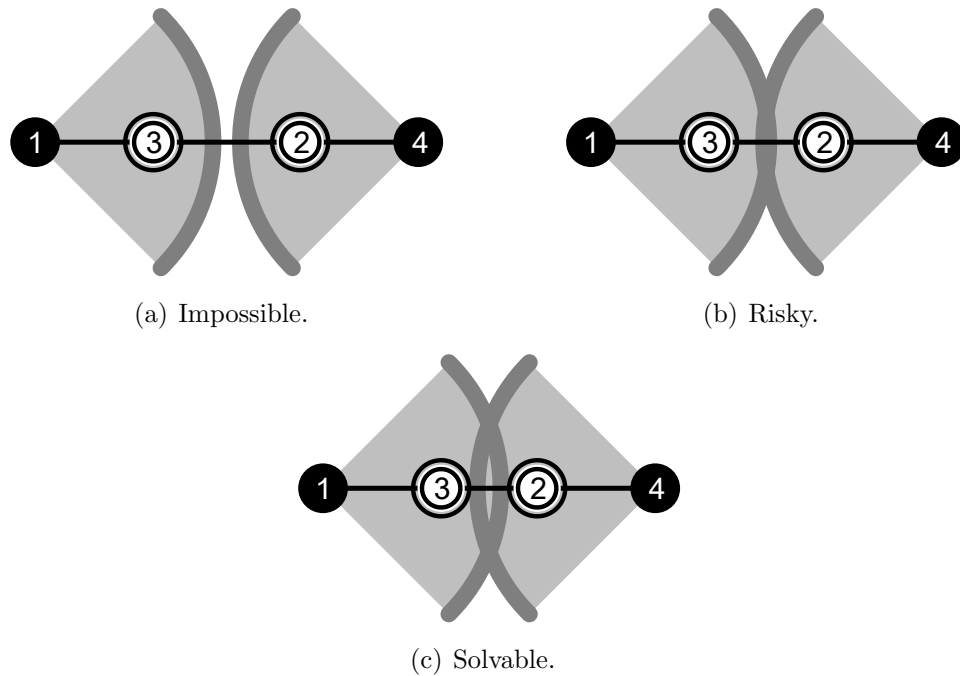


Figure 5.5 : **a:** There is no way to solve this problem, for there is no range overlap between robot 1 and robot 4, thus robot 2 and robot 3 cannot safely swap positions. **b:** It is too risky for robot 2 and robot 3 to make a swap, since unsafe ranges overlap but safe ranges do not overlap. They will stop at the boundary between safe range and unsafe range. **c:** This problem is solvable because of safe range overlapping. Safety feature prevents one normal robot leaving the safe range overlap before another robot enters it.

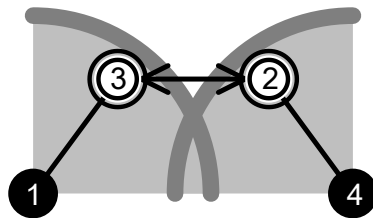


Figure 5.6 : Robot 2 and robot 3 are local minimum and maximum. They move to each other and cause disconnection. Range-based safety causes deadlock and cannot solve this problem.

## 5.4 Experimental Results

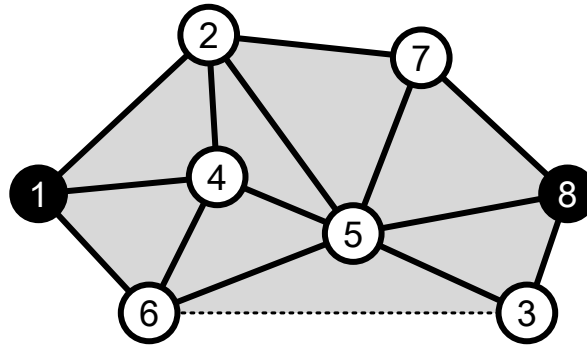
I have performed several experiments on my simulation software, and also on our robot platform, the r-one robot. The simulation software uses a continuous sensor model, a departure from the discrete low-resolution sensor on the robot, to provide accurate results, as well as be able to simulate discrete angle measurement. It also allows us to model some physical effects, such as errors, collisions, and communication obstructions.

### 5.4.1 Simulation

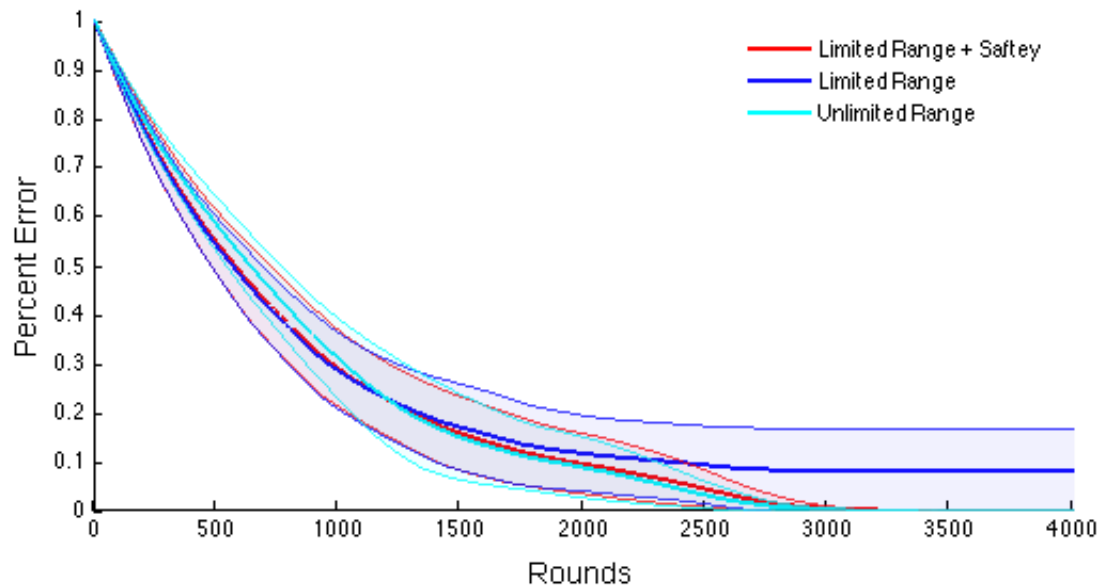
My simulation software implements sensing errors and communication interference. The simulation software supports up to 100 robots, and updates all movements and communications every 20 milliseconds. To analyze the process of sorting robots, we measure the area of convex hull of the robot configuration (Fig.5.7(a)), where the convex hull is the minimum polygon that covers all the robots. Thus, all the vertices of convex hull are robots, and robots are divided into two groups: convex hull vertices and non-vertices.

For each vertex robot, all other robots appear in a sector smaller than 180 degrees. Since we have only two kinds of movements: moving to midpoint (or bisector of two robots), and moving to parent (one robot), there is no movement for a vertex robot to move outward the convex hull. Thus, area of convex hull cannot increase. If area of the convex hull becomes zero, all robots must be in a line.

Fig.5.7(b) shows the effect of range-based safety. Limited range without safety has a large possibility to fail (area of convex hull does not converge to zero), and range-based safety prevented failure without significantly losing efficiency.



(a) Convex hull of robots.



(b) Area of convex hull (simulation).

Figure 5.7 : **a**: Convex hull of the robot configuration is shown in grey. Our distributed controller always reduces the area of the convex hull, which converges to zero if robots are in a line. **b**: Area of convex hull versus time for the same initial configuration with three different simulation mode: unsafe (limited range), safe (limited range with range-based safety), and unlimited range.

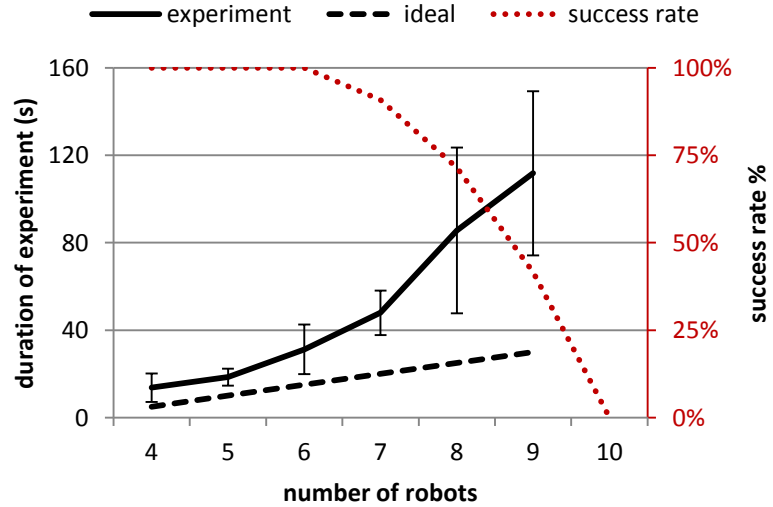


Figure 5.8 : Physical sorting with eight robots. Collision and interference happened, but robots managed to get sorted.

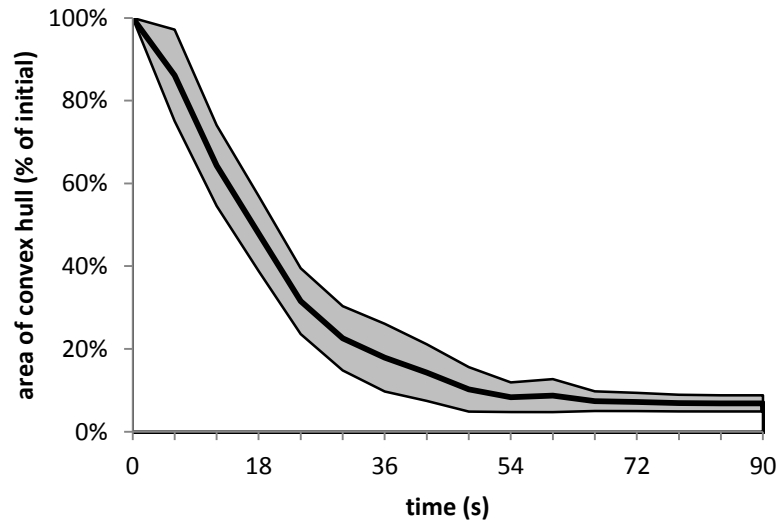
### 5.4.2 Hardware

We have performed several hardware experiments, using the r-one robots shown in [34]. The capability of this platform supports the assumptions in our problem statement; each robot can measure the bearings to its nearby robots, however the hardware has a limited resolution of only  $\frac{\pi}{8}$ . We implemented an algorithm using inter-robot communication. Distance measurement has an accuracy of about 100mm within 1,200mm. Each robot also has eight bump sensors that provide obstacle detection. Rounds on robots are not synchronized, but a message buffer refresh and timeout schedule keeps them running as if they are synchronized [15].

In my simulation I ignore collision and allow robots to overlap, or treat a robot as a single point. In the real world, robots will collide with each other. There is an obstacle avoidance behavior that can effectively maneuver the robot away from other robots. Collisions make the problem difficult. When robot collides to each other, it will suspend the algorithm, back up a few centimeters, rotate some angle, try to move forward a few centimeters, and continue the algorithm. This may cause the robot to move out of current convex hull, thus increase the convex hull area. Furthermore, congestion may cause robots to be trapped and crowded together. In addition, if the distance between global minimum and maximum is not big enough to place all other



(a) Time versus number of robots.



(b) Area of convex hull (hardware).

Figure 5.9 : **a**: Time spent versus number of robots. For each number of robots, 10 experiments with the same initial configuration are performed, and mean and standard deviation of time spent are shown in the figure. When number of robots increases, collision and interference become a severe problem and cause failure. **b**: Area of convex hull among time in one experiment. It decreases as robots become sorted, while physical interference may cause it to increase. 7% of the area cannot be removed due to sensor resolution and measurement error.

robots, the problem becomes unsolvable. Photo of experiment is shown in Fig.5.8.

To evaluate a sorting quality over time, I use the AprilTag system by APRIL group [47]. This measures the ground truth position,  $P_u = \{x_u, y_u, \theta_u\}$ , of each robot  $u$ . Area of convex hull can be monitored on a computer in real time, and graphs can be generated as shown in Fig.5.9. The robots are not allowed to use the ground-truth position while executing our algorithms.

## Chapter 6

### Topology-based Sorting Algorithm

In the straightforward geometry-based sorting algorithm, I tried to make patches to solve issues, but new issues appear and are difficult to solve. I need a completely new algorithm to solve the sorting problem. Topology could give robots more information than finding a predecessor or successor, and using the topology information in an asymmetric way leads to the new topology-based sorting algorithm.

#### 6.1 Algorithm

My goal is to form a topologically sorted path amongst all robots, while arranging them into a geometrically straight and evenly distributed line (6.1). I accomplish this with a set of atomic *topological operations* that are fully distributed and do not require mutual exclusion between robots.

Uninitialized robots join the *primary tree* and the *feedback tree*. Initially, all the robots are uninitialized, and the process of joining these trees builds a *main path*. The main path is initially the shortest path between global minimum and global maximum, and it may not be sorted. I apply topological operations to modify the main path: Expansion and trimming add and remove edges, so that I can enlarge sorted parts and eliminate unsorted parts on the main path; Navigation along the main path changes local topology to move out-of-order robots to their correct positions. In addition, I perform geometric operations to straighten the main path, and bring other robots



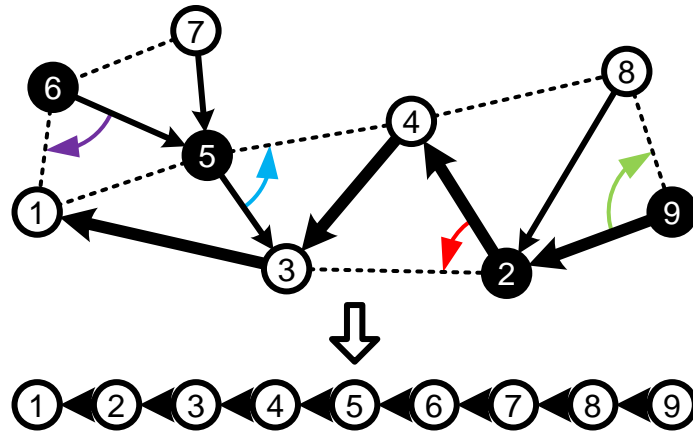


Figure 6.1 : Topology-based sorting.

closer to the main path. All these operations run in parallel on each robot. I describe these operations in this section, and present analysis in the next section.

### 6.1.1 Main Path Formation

The *main path* (Fig.6.2) always connects the global minimum and the global maximum. I construct it using two trees: the primary tree and the feedback tree. The *primary tree* is a spanning tree rooted at the global minimum. This tree is constructed using distributed BFS broadcast messages, and a leader election protocol roots it at the robot with the lowest label [15]. Each robot stores label of its parent, while global minimum sets it to null. At time  $t = 0$ , robot  $u$  has no information about neighbors, so it assumes itself as global minimum, and broadcast this to its neighbors. After the first round, it receives messages from neighbors. If it finds another robot with lower label, it stops considering itself as global minimum, and relays the message initially from the source with lowest label. After at most  $n$  rounds, only the robot with the lowest label in the graph still considers itself as the global minimum, and each other

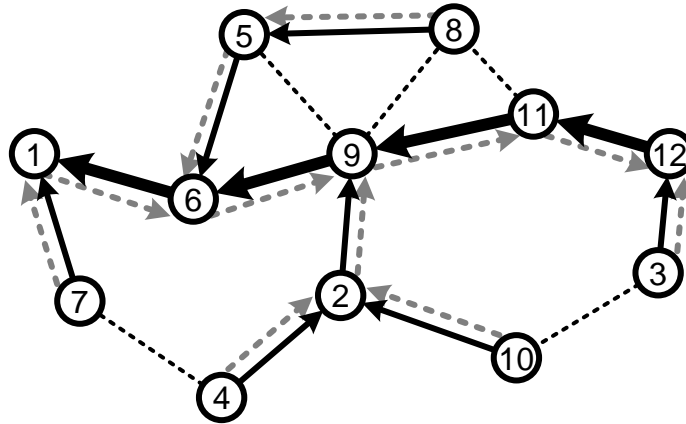


Figure 6.2 : The primary tree drawn in solid arrows and the feedback tree drawn in dashed arrows. Arrow from child to parent means child stores parent’s label, while information flows in the opposite direction. The main path is highlighted as bold arrows. Neighbor connections not used to form trees are drawn with thin dashed lines.

robot knows the existence of the global minimum and selects a parent toward it.

Once the primary tree is built, it does not automatically refresh to produce the shortest paths between the root and the leaves. Once robots select parents they keep them unless they are modified by one of the topological operations described later in this section, or under four critical changes in tree topology:

- robot has no parent (initial case),
- loses connection to previous parent,
- global minimum disconnected, or
- new global minimum discovered.

If any of these happens, robot selects parent among its neighbors with the fewest hops.

The *feedback tree* is another spanning tree overlaying the primary tree. All robots initially assume themselves as global maximum, and after at most  $n$  rounds of communication, only the robot with the highest label still considers itself as the global maximum, and each other robot knows the global maximum and selects a parent toward it, and this forms the feedback tree. Only the edges in the primary tree are used for the feedback tree, that is, robot  $u$  only select robot  $v$  as parent of feedback tree, if  $u$  is  $v$ 's parent or  $v$  is  $u$ 's parent in the primary tree. In the feedback tree, robot has no preference with previous parent, and refreshes the feedback tree to make it consistent with the primary tree.

The feedback tree provides a feedback from global maximum to global minimum, so that:

- robots on the main path (the path in both trees connecting global minimum and global maximum) know that they are on the main path, because parent of primary tree and parent of feedback tree are different, and
- robots not on the main path (they form *sub-trees*) also know, because parent of primary tree and parent of feedback tree are the same.

In order to build primary tree and feedback tree, each robot  $u$  broadcasts the following information to its neighbors in each round, which is also essential to apply operations in the remaining subsections:

- $u.id$ ,
- $u.primary.parent, u.primary.source, u.primary.hop$ ,
- $u.feedback.parent, u.feedback.source, u.feedback.hop$ ,
- $u.status$ ,

where status means whether on the main path. They do not need to share any measurements, decisions, or lists of neighbors with the network.

Now we have the main path, and several sub-trees attached to the main path. Then our goal is clear:

- change the topology of the primary tree into a path, which is to join all robots onto the main path and eliminate all sub-trees,
- straighten the main path and get robots evenly distributed on the main path, and
- sort robots on the main path to get the correct order.

Since feedback tree only provides feedback for robots to detect whether they are on the main path, and is refreshed in every round, our topological operations will only be based on the primary tree. This is different from our previous symmetric geometry-based algorithm [50], since two trees (and topological operations described below) are asymmetric between minimum and maximum. In the remainder of this paper, when I talk about parent, child, or sibling, I refer to the primary tree unless noted.

### 6.1.2 Geometric Operations (Robot motion)

With the primary tree and the feedback tree built, all robots will know whether they are on the main path. They have different geometric behaviors.

- For global minimum or global maximum, they remain stationary to let others fit between them.

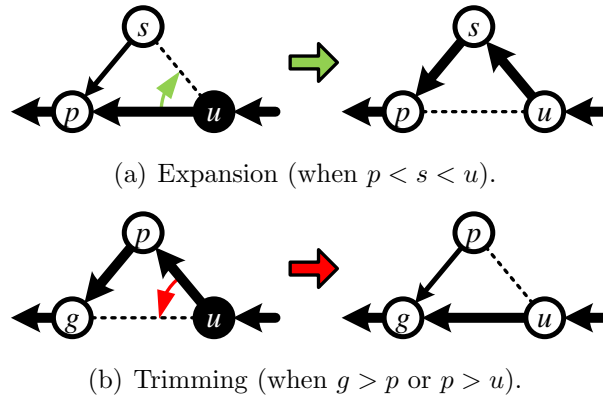


Figure 6.3 : Main path operations.

- For each other robot on the main path, it moves to midpoint between parent of primary tree and parent of feedback tree. This straightens the main path and makes robots on the main path evenly distributed.
- For each robot in sub-trees, parents of both trees are the same. It moves toward its parent. This causes sub-trees to collapse, and all robots move toward the main path.

These moving behaviors make geometric changes to the network, and since scattered robots move closer to become neighbors, geometric changes may result in topological changes.

### 6.1.3 Main Path Topology Operations

There are two kinds of main path operations: expansion and trimming (Fig.6.3). If robot  $u$  on the main path discovers one of these situations, it reselects parent to change the topology of the main path.

*Expansion* adds one robot onto the main path (Fig.6.3(a)). If local robot  $u$  is on the main path, and there exist neighbors  $p$  and  $s$  such that:

- $p$  is on main path and is  $u$ 's parent,
- $s$  also selects  $p$  as parent ( $s$  must be a common neighbor and  $u$ 's sibling in sub-tree), and
- $p < s < u$ ,

then  $u$  reselects  $s$  as parent instead of current parent  $p$ . Expansion inserts robot with correct local order onto the main path. If there are more than one robots that can be selected as  $s$ ,  $u$  can simply pick one of them.

*Trimming* removes one robot from the main path (Fig.6.3(b)). If local robot  $u$  is on the main path, and there exist neighbors  $p$  and  $g$  such that:

- $p$  is on main path and is  $u$ 's parent,
- $g$  is  $p$ 's parent ( $u$ 's grandparent, must also be on the main path), and
- $g > p$  or  $p > u$ ,

then  $u$  reselects  $g$  as parent instead of current parent  $p$ . Trimming removes locally disordered robot out of main path while remain connected. Those robots trimmed out will finally get back onto the main path at appropriate positions with navigation (described below) followed by an expansion.

#### 6.1.4 Sub-tree Topology Operations

Robots in sub-trees also have operations to change sub-tree topology, so that they will move (topologically and geometrically) to appropriate positions to get expanded onto the main path. Navigation to minimum and maximum (Fig.6.4) allows sub-trees to move along the main path, and seek for positions to be expanded.

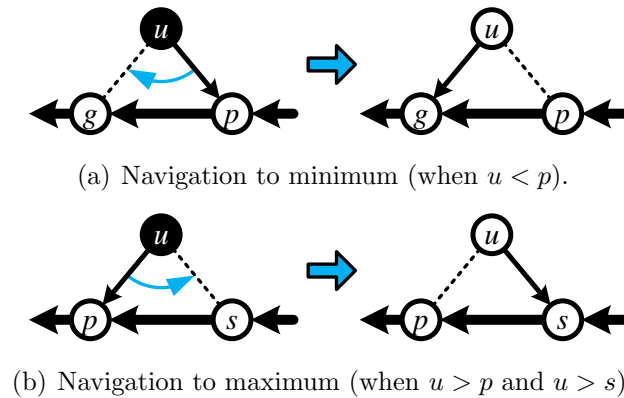


Figure 6.4 : Sub-tree operations.

*Navigation to minimum* moves sub-tree robot toward global minimum if it is smaller than its parent (Fig.6.4(a)). If local robot  $u$  is in sub-tree, and there exist neighbors  $p$  and  $g$  such that:

- $p$  is on the main path and is  $u$ 's parent,
- $g$  is  $p$ 's parent ( $u$ 's grandparent, must also be on the main path), and
- $u < p$ ,

then  $u$  reselects  $g$  as parent instead of current parent  $p$ .

This operation changes sub-tree topology to let low-label robot get closer to its grandparent, when it has lower label than its parent. If  $g < p < u$ , it will get expanded onto the main path. Otherwise, this process repeats after  $u$  geometrically moves toward  $g$  to become neighbor of  $g$ 's parent. This operation also eliminates all sub-trees connected to global maximum, as they cannot get expanded since they do not have a sibling on the main path.

*Navigation to maximum* moves sub-tree robot toward global maximum if it is greater than both its parent and its main-path sibling (Fig.6.4(b)). If local robot  $u$

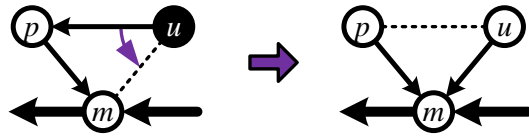


Figure 6.5 : Sub-tree collapse.  $m$  could be any robot on the main path (not necessarily  $p$ 's parent), and  $p$  could be any hops away from the main path.

is in sub-tree, and there exist neighbors  $p$  and  $s$  such that:

- $p$  is on main path and is  $u$ 's parent,
- $s$  is also on main path and  $s$  selects  $p$  as parent ( $s$  is  $u$ 's sibling), and
- $u > p$  and  $u > s$ ,

then  $u$  reselects  $s$  as parent instead of current parent  $p$ .

This operation changes sub-tree topology to let high-label robot move towards its sibling on the main path.  $u$  continues to move toward global maximum until it has higher label than its sibling on the main path. Finally it will have another parent  $p'$  and sibling  $s'$  with  $p' < u < s'$ , and sibling  $s'$  will perform an expansion to insert  $u$  onto the main path.

An optional sub-tree collapse reduces depth of sub-trees to accelerate navigation (Fig.6.5). If local robot  $u$  is in sub-tree, and there exist neighbors  $p$  and  $m$  such that:

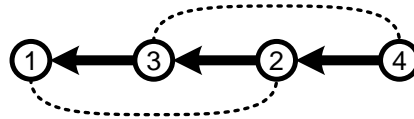
- $p$  is in sub-tree and is  $u$ 's parent, and
- $m$  is any robot on the main path,

then  $u$  reselects  $m$  as parent instead of current parent  $p$ . This operation avoids the whole sub-trees move back and forth while navigating, since there are no other topological operations for robots more than one hop from the main path. If there are more than one robots that can be selected as  $m$ ,  $u$  can simply pick one of them.





(a) Unsortable.



(b) Sortable after global maximum moved.

Figure 6.6 : **a:** This configuration cannot be solved unless we shorten the main path to gain at least one more connection. **b:** Sortable after global maximum moved (degenerated triangles shown with curved edges). Detailed solving process will be shown in Fig.6.8.

### 6.1.5 Movement of Endpoints

In some cases robots are far away from each other so that no operations can be made on the main path (Fig.6.6(a)). Robots can still form a line and get evenly distributed by this algorithm, but not sorted. In order to solve this problem, we can allow global maximum to move. If the global maximum cannot directly contact with its grandparent, it moves toward its parent, until its grandparent becomes a neighbor. Applying geometric operation move-to-midpoint at the same time, eventually all robots on the main path can have connection of at least two hops along the main path (Fig.6.1-Sortable)\*, which enables them to trim disordered robots out.

In most cases, we only care about the position of global minimum, which becomes the head of the queue, but not global maximum as the tail. Global maximum movement also enables us to drive the global minimum, and other robots will get sorted behind and follows the global minimum. This enables robots to get sorted while they

---

\*Configurations with fewer than three robots are trivial cases. We can require the global maximum to see its grandgrandparent (three hops away) to enforce all other main-path robots having two-hop connections.

are on their way back home, without stopping and spending more time to sort before recovery.

On the other hand, if we care about overcrowding, global maximum can carefully move away from its parent when it has too many main path neighbors. This will stretch the main path to reduce degree of connection to eliminate overcrowding.

## 6.2 Analysis

In this section we prove that this algorithm is safe, correct, and effective.

### 6.2.1 Safety

Safety is the most important issue for sorting robots. Once the connected network breaks into two or more parts, robots cannot talk with each other and algorithm will fail, since no distributed algorithm works with unconnected network. Some critical connections must be maintained while robots move to keep connectivity. Our old symmetric algorithm [50] is unsafe: local extrema in that algorithm is very likely to tear the network apart, and we even provided an example that only applying move-to-midpoint could left someone behind. With a range-based safety constrain, that algorithm could still cause deadlock. But in our new asymmetric algorithm, we can prove it is intrinsically safe and will not cause disconnection.

*All topological operations are safe.* We begin with a theorem in graph theory that if an undirected graph has  $n$  nodes and  $n - 1$  edges, and there are no cycles, the graph must be connected. We have  $n$  robots, with  $n - 1$  child-to-parent connections associated to them (global minimum has no parent), assuming that communication is bidirectional, and then we fall in this case. Therefore, if the graph breaks, there must be a cycle.

Here we show that any combinations of operations are safe, even if some robots make competing decisions at the same time. We have only four topological operations (expansion, trimming, navigation, and collapse), and none of them could directly select a descendent as parent. If multiple robots make decisions and perform operations on their own judgment, a robot being selected as parent could reselect its parent at the same time. Trimming and navigation to minimum always reselect parent with fewer hops to root, therefore they could not be responsible for generating cycles. Expansion and navigation to maximum reselect parent with more hops, but our algorithm guarantees that they will not generate cycles:

- Concurrent multiple expansions do not influence with each other, since they can only happen between different pairs of parent and child on the main path.
- Concurrent multiple navigations to maximum do not influence with each other, since topology of the main path does not change, and navigating robots do not select each other as parent since they are in sub-trees.
- Concurrent expansion and navigation to maximum: expansion requires the sibling on the main path has greater ID, while navigation to maximum requires the sibling on the main path has lower ID.

Therefore, all topological operations are safe.

*All geometric operations are safe.* There are only two kinds of geometric operations: move to midpoint of two robots, and move towards one robot. Here we prove that neither of them causes disconnection.

- Only robots on the main path move to midpoint of two other robots, and they form a chain. As proved by other works [24], distance between pairwise parent and child cannot increase, therefore the main path cannot break.

- Only robots in sub-trees move toward another robot. If a robot chases another with equal or greater speed, distance between them cannot increase, therefore sub-trees cannot break [25]. We can also employ recovery algorithms [33] to enhance safety.
- Global maximum moving toward its parent cannot move it outside the boundary set by [25], thus it cannot cause disconnection. If we allow global maximum to move away from its parent to avoid overcrowding, and since we require three or more main-path neighbors, we still stay in the boundary.
- If we allow global minimum to move, it may break the connection between its child on the main path and itself, since global maximum may not be pushing fast enough. Therefore, we need the global minimum to drive carefully, and slow down if there is a risk to break the connection. Applying safety range or requiring connection with grandchild could simply solve this problem.

Therefore, all geometric operations are safe.

### 6.2.2 Correctness

*First, we show that there are no loops of operations causing deadlock.* All topological operations (except sub-tree collapse) in our algorithm are based on triangles adjacent to the main path, which has one or two edges on the main path. (They can be geometrically degenerated triangles.) We can enumerate that there are only three different triangle topologies. Three robots have relative orders, but there are only six permutations of three different integers. Therefore, there are only eighteen combinations and we can enumerate them (see Fig.6.7), and show which operations are applied to change the topology. Unstable triangles will become *topologically stable*

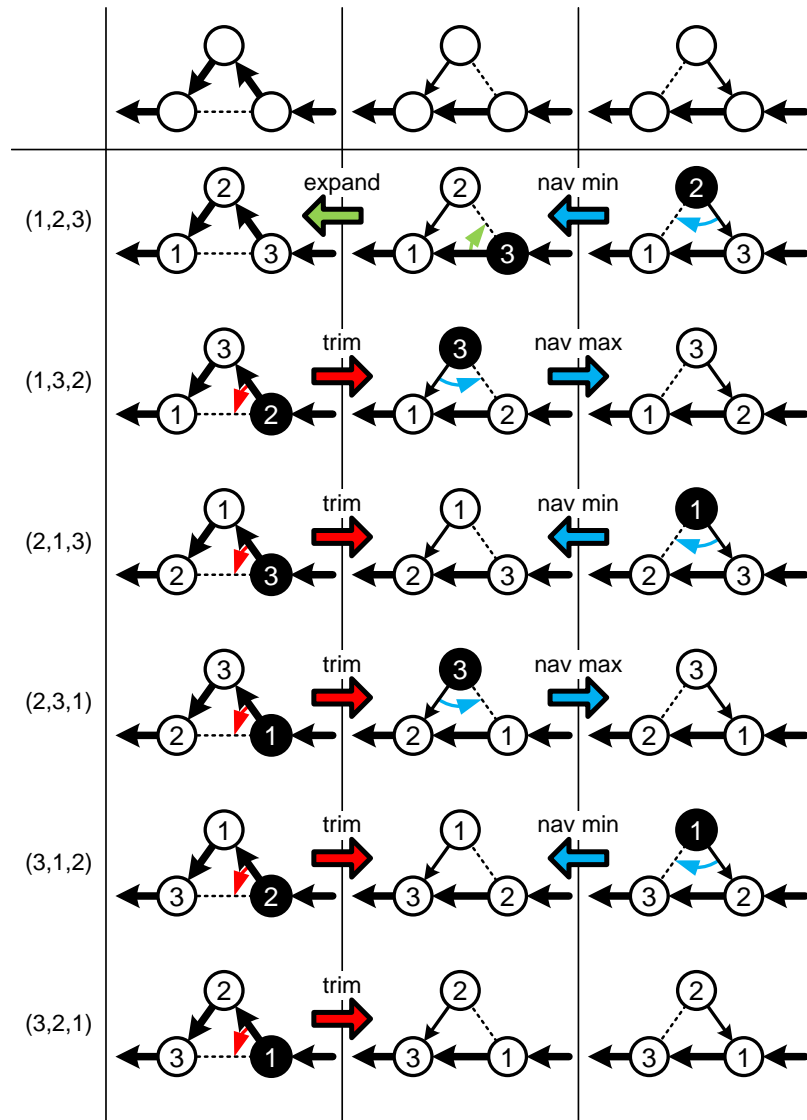


Figure 6.7 : IDs of three robots forming a triangle have relative order, and we denoted 1, 2, and 3 as their ascending order. There are only six permutations of three different numbers and three different topologies for triangles adjacent to the main path, so we have eighteen local triangle states. There are no loops of operations between any states.

after one or two operations, where stable means no more topological operations can be applied in this triangle to revert it into previous topology.

We also concern about concurrent operations in nearby triangles. We can enumerate combinations of triangles as follows.

- Concurrent multiple expansions. Multiple robots can be expanded onto the main path at different positions, but only one can get expanded at the same position in one round. Once expanded, it cannot be trimmed out unless its parent or child is trimmed out.
- Concurrent multiple trimmings. A robot being trimmed out may also trim out its parent at the same time. As we mentioned in previous subsection, the main path is still connected and multi-hop sub-trees may be generated. Robots in sub-trees will navigate, and get expanded in other positions. They will not get expanded at their previous positions.
- Concurrent multiple navigations. Navigations on different robots are performed independently and do not change topology of the main path. Navigating to minimum and to maximum for the same robot is exclusive, so a robot cannot navigate back and forth between two positions.
- Expansion and trimming. A robot being trimmed out may expand another robot between itself and its parent. This also generates multi-hop sub-tree, and also cannot be reverted once trimmed. A robot being expanded is not already on the main path to trim another robot out.
- Trimming and navigation. A robot may navigate to some robot that is being trimmed out. This also generates multi-hop sub-tree, and also cannot be re-

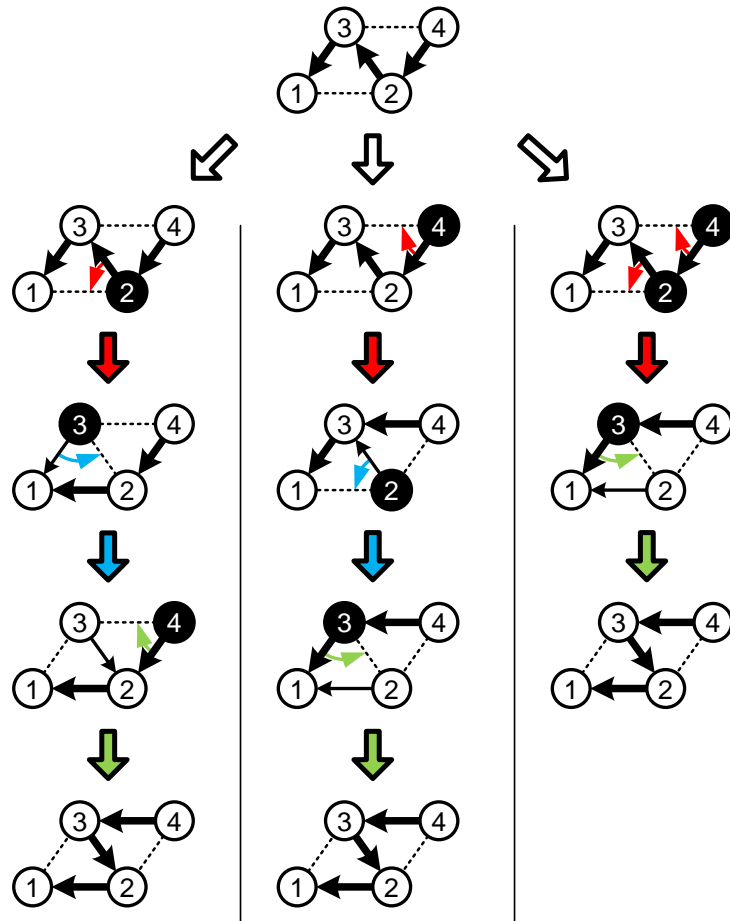


Figure 6.8 : Robot 2 and robot 4 may perform operations sequentially or concurrently. All three cases will end in correctly sorted main path.

verted once trimmed. A robot performing navigation is not on the main path to trim another robot out.

- Navigation and expansion. A robot being expanded cannot navigate: conditions of navigation and expansion are exclusive. A robot performing navigation is not on the main path to expand another robot.

Therefore, even concurrency happens, this algorithm will not generate loop of topology changes. We show an example that different sequences of operations turn

into the same correct result (Fig.6.8).

*Second, we show that all robots will be expanded onto the main path.* If the algorithm terminates with one or more robots in sub-trees, there must be at least one robot  $u$  having a parent on the main path. Geometric operation should have moved  $u$  toward its parent  $p$ , in order to see its grandparent  $g$  and main-path sibling  $s$  (if exist). If it has both parent  $p$  and main-path sibling  $s$ , and  $p < u < s$ , it will get expanded. Otherwise, it must either reselect  $g$  (if  $u < p$ ) or  $s$  (if  $u > p$  and  $u > s$ ) as parent and move toward it. Those comparisons are both inclusive and exclusive, therefore an operation must be performed immediately. Contradiction.

Each robot in sub-trees will finally reach a position to get expanded, and that position always exists. If  $u < p$  it navigates to minimum, there must exist pair of robots  $(p', s')$  on the way to global minimum such that  $p' < u < s'$ , because global minimum is lower than  $u$ . If  $u > p$  and  $u > s$  it navigates to maximum, there must exist pair of robots  $(p', s')$  on the way to global maximum such that  $p' < u < s'$ , because global maximum is greater than  $u$ . In both cases an expansion will be performed.

*Third, we show that the main path will be sorted in ascending order.* If we do not allow global maximum to move, this algorithm may terminate unsorted because there are no triangles to perform operations (Fig.6.6(a)). Allowing global maximum moving toward its parent will compress the main path, and finally all robots on the main path can directly communicate with their grandparents and grandchildren, it exist. So we assume there always exist triangles with two edges on the main path at any possible positions.

Order of IDs on the main path form a integer sequence. It starts from the global minimum, contains one or more ascending segments and zero or more descending



segments, and ends at the global maximum. Descending segments should be avoided, and once eliminated, the main path is sorted. Here we prove that the algorithm removes all descending segments.

We present two metrics to measure the disorder:

- Number of reverses. For example, in  $(1,7,5,2,3,6,4,8)$  three robots  $\{5,2,4\}$  are smaller than parents on their left.
- Total dropping. For the same example,  $(7 - 2) + (6 - 4) = 7$ .

From any initial case, those two metrics are finite natural numbers. No descending segment exists if and only if either of them is zero, and they must be both zero or non-zero. In our algorithm, neither of these metrics could increase:

Expansion does not create or enlarge any descending segments, for we require  $p < u < s$ . Navigation and collapse do not change the main path. For trimming, we have two situations:

- The robot being trimmed is at the boundary of a descending segment and an ascending segment. Total dropping must decrease, since dropping of this segment must decrease as local extremum is removed. Number of reverses may decrease or remain the same. For the example above, removing 2 changes descending segment  $(7,5,2)$  into  $(7,5,3)$  and reduces dropping.
- The robot being trimmed is not at the boundary. Total dropping remains the same, but number of reverses must decrease. For the example above, removing 5 eliminated one reversing item.

As we assumed triangles adjacent to the main path always exist, there always exists trimming operation if there exists local extremum. Each trimming either reduces

total dropping or reduces number of reverses. Since they are natural numbers, they must reduce to zero. Therefore all descending segments will be removed and leave the main path a single ascending segment.

*Fourth, we show that all robots on the main path will form a line and distribute evenly.* Since each robot on the main path only moves toward midpoint between its parent and child on the main path, they form a chain, and many previous works [24] [25] have proved that the chain will be straightened and robots get evenly distributed along the chain.

As a consequence, this algorithm is correct in all circumstances. It is also robust to any critical changes caused by robot failure or human intervention (if the network is still connected), since the main path rebuilds in these cases.

### 6.2.3 Effectiveness

We show that time spent and distance traveled on each robot are bounded by  $O(n^2)$ .

First, we assume all robots are either on the main path or only one hop away from the main path. There are  $k$  ascending segments and  $k - 1$  descending segments on the main path, and we describe the  $i$ th ascending segment as  $(s_i, t_i)$ , where  $s_i$  and  $t_i$  are boundary robots (local extrema). For a robot  $u$ , there must exist at least one ascending segment in which  $u$  falls in the range, i.e.  $s_i \leq u \leq t_i$ .  $u$  may navigate to join one of them, but once joined,  $u$  will not get trimmed out until  $u$  becomes boundary of that segment. If  $u$  is at the boundary and get trimmed out,  $u$  no longer falls in the range of this ascending segment. Range of ascending segments cannot increase, for we only expand inside but not at the boundary. Two ascending segments  $(s_i, t_i)$  and  $(s_{i+1}, t_{i+1})$  separated by a descending segment can merge (by trimming all robots in the descending segment out), and the new ascending segment

will be  $(s_i, t_{i+1})$ .  $u$  falls in  $(s_i, t_{i+1})$  only if  $u$  falls in  $(s_i, t_i)$  and/or  $u$  falls in  $(s_{i+1}, t_{i+1})$ , because  $t_i > s_{i+1}$ . Thus,  $u$  can only get trimmed and expanded at most  $k$  times. Since  $k$  can be proportional to  $n$  and each navigation travels at most  $n - 2$  hops, distance traveled for navigation is bounded by  $O(n^2)$ .

Second, a robot may become two hops away from the main path, for its parent may be trimmed out. It travels constant distance to get back to one hop, and this can happen at most  $n - 3$  times. Thus, it takes  $O(n)$  total distance for each robot to get back into one hop away from the main path.

Third, it takes up to  $O(n)$  to straighten the main path, for distance between any robots should fall in  $n \times r_{\max}$ .

Fourth, multi-hop sub-tree collapse takes at most  $n - 2$  hops from any leaf to the main path, so it takes  $O(n)$  distance for a robot to get within one hop to the main path.

Therefore, worst case distance complexity on each robot is  $O(n^2)$ , if we manually control the sequence of operations. Movements are optimized in this algorithm than previous work [50]. Thus, when multiple operations are performed at the same time, robots do not have to navigate back and forth, so we expect  $O(n)$  in average case.

Movements are optimized in this algorithm than previous work. Sub-trees collapse to the main path, and once collapsed, robots stick with the main path to avoid unnecessary sideways movements. Thus, when multiple operations are performed at the same time, robots do not have to navigate back and forth, so we expect  $O(n)$  in average case.

Since robots do not need to share lists of neighbors, message size is fixed. The system only requires  $n$  local broadcast messages to be sent at each round. Our algorithm can be carefully implemented such that a robot only need to go through

its list of neighbors once to make decision, messages can be discarded after received and processed, so each robots requires  $O(1)$  size of memory regardless the size of network (provided we have enough communication bandwidth). This also enables this algorithm to work on large quantity of low-cost robots.

### 6.3 Implementation

---

**Algorithm 4** SORTING( $u$ )
 

---

```

1: PRIMARYTREE( $u$ )
2: FEEDBACKTREE( $u$ )
3: if  $u.primary.parent \neq u.feedback.parent$  then
4:    $u.status \leftarrow on\_main\_path$ 
5:   for each robot  $v$  in  $u.neighbors$  do
6:     EXPANSION( $u, v$ )
7:     TRIMMING( $u, v$ )
8:   end for
9:   if  $u.primary.parent \neq \emptyset$  and  $u.feedback.parent \neq \emptyset$  then
10:    MOVETOMIDPOINT( $u.primary.parent, u.feedback.parent$ )
11:   end if
12: else
13:    $u.status \leftarrow in\_sub-trees$ 
14:   for each robot  $v$  in  $u.neighbors$  do
15:     NAVIGATION( $u, v$ )
16:     COLLAPSE( $u, v$ )
17:   end for
18:   MOVETOWARD( $u.primary.parent$ )
19: end if

```

---

The algorithm needs to enumerate all messages received in this round, decide whether itself is on the main path, see if any topological operation can be done (pick one if multiple topological operations available), and perform geometric operation. A straight-forward version of the main algorithm (Algorithm 4) requires processing the neighbor messages multiple times, in order to update the trees and perform opera-

tions. This requires robot having  $O(n)$  memory to store the list of neighbors.

Operations are described in Algorithm 5, where  $u.parent$  refers to  $u.primary.parent$ , and comparing robots means comparing their IDs. Topological operations check requirements and decide whether to perform or not. If multiple operations are qualified to be performed in the same round, robot simply chooses one.

A modified version for  $O(1)$  size of memory is also provided (Algorithm 6), if robot can process each message promptly and discard it after receiving another constant number of messages. Robot makes decision based on previous tree structure, and stores a preferred topological operation if possible. If tree structure changes in this round, change of tree structure is committed and topology operation is discarded. Then, if there is a topological operation stored, it performs that operation. Robot stops if tree changes or topological operation selected, and geometric operation is performed only when nothing changed. Message from parent is cached so that we can always know whether a neighbor is grandparent or sibling.

## 6.4 Experimental Results

We have performed several experiments on our simulation software. Our simulation software supports up to 100 robots, and updates all movements and communications every 20 milliseconds. The simulation software can provide absolutely correct sensor readings and tree structures, as well as modeling some physical effects, such as errors, collisions, obstructions, and communication delay.

To analyze the process of sorting robots, we measure the area of convex hull of the robot configuration. For each vertex robot, all other robots appear in a sector smaller than 180 degrees. Since we have only two kinds of movements, moving to midpoint and moving to parent, there is no movement for a vertex robot to move outward the

---

**Algorithm 5 OPERATIONS**


---

```

1: function PRIMARYTREE( $u$ )
2:   if  $u.primary.parent = \emptyset$  or ( $u.primary.parent$  not in  $u.neighbors$ ) or (global
   minimum changed) then
3:      $u.primary.parent \leftarrow v \in u.neighbors$  with fewest hops to global minimum
4:   end if
5: end function
6: function FEEDBACK( $u$ )
7:   for each robot  $v$  in  $u.neighbors$  do
8:     if ( $u.primary.parent = v$  or  $v.primary.parent = u$ ) and ( $v$  has fewest
     hops to global maximum) then
9:        $u.feedback.parent \leftarrow v$ 
10:    end if
11:  end for
12: end function
13: function EXPANSION( $u, v$ )
14:  if  $u.parent = v.parent$  and  $u.parent < v < u$  then
15:     $u.parent \leftarrow v$ 
16:  end if
17: end function
18: function TRIMMING( $u, v$ )
19:  if  $u.parent.parent = v$  and ( $v > u.parent$  or  $u.parent > v$ ) then
20:     $u.parent \leftarrow v$ 
21:  end if
22: end function
23: function NAVIGATION( $u, v$ )
24:  if  $u.parent.parent = v$  and  $u < u.parent$  then
25:     $u.parent \leftarrow v$ 
26:  end if
27:  if  $u.parent = v.parent$  and  $v.status = on\_main\_path$  and  $u > u.parent$  and
    $u > v$  then
28:     $u.parent \leftarrow v$ 
29:  end if
30: end function
31: function COLLAPSE( $u, v$ )
32:  if ( $u.parent.status = in\_sub-trees$ ) and  $v.status = on\_main\_path$  then
33:     $u.parent \leftarrow v$ 
34:  end if
35: end function

```

---

---

**Algorithm 6** SORTING WITH CONSTANT MEMORY
 

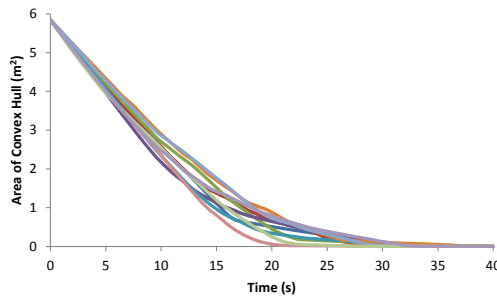
---

```

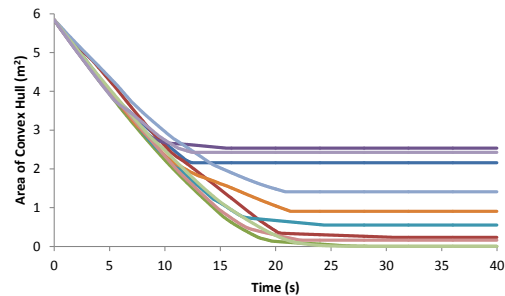
1: function RECEIVEMESSAGEHANDLER( $u, v$ )
2:   if  $u.status = on\_main\_path$  then
3:      $u.action \leftarrow EXPANSION(u, v)$ 
4:      $u.action \leftarrow TRIMMING(u, v)$ 
5:   else
6:      $u.action \leftarrow NAVIGATION(u, v)$ 
7:      $u.action \leftarrow COLLAPSE(u, v)$ 
8:   end if
9:    $u.tree = \text{calculate optimal primary tree and feedback tree}$ 
10: end function
11: function ROUNDTIMERHANDLER( $u$ )
12:   if  $u.tree$  different from current trees then
13:     commit  $u.tree$ 
14:     update  $u.status$ 
15:     cache  $u.parent$ 
16:      $u.action \leftarrow \emptyset$ 
17:   else if  $u.action \neq \emptyset$  then
18:     commit  $u.action$ 
19:   else if  $u.status = on\_main\_path$  then
20:      $MOVE\_TO\_MIDPOINT(u.primary.parent, u.feedback.parent)$ 
21:   else
22:      $MOVE\_TOWARD(u.primary.parent)$ 
23:   end if
24: end function

```

---

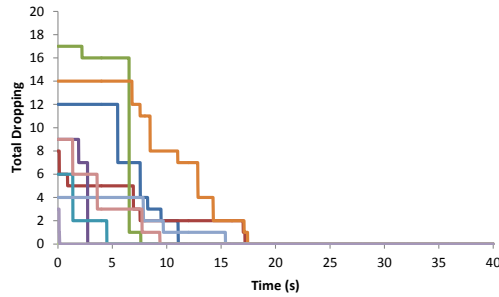


(a) Topology-based (new) algorithm.

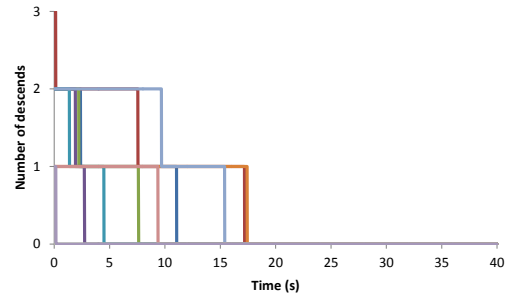


(b) Geometry-based (old) algorithm.

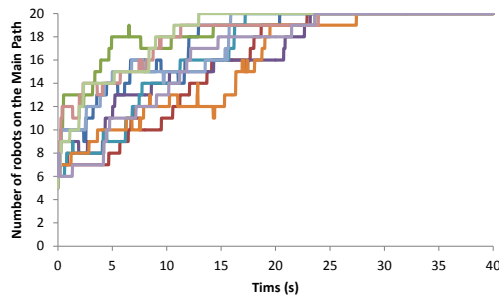
Figure 6.9 : Area of convex hull versus time. Each of ten configurations shown in different color is done with both algorithms.



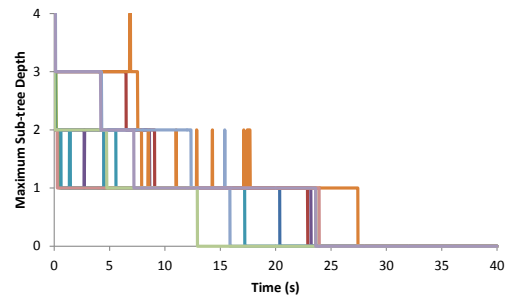
(a) Total dropping always decreases.



(b) Number of reverses always 64-PlotDecreases.



(c) Number of robots on the main path.



(d) Maximum sub-tree depth.

Figure 6.10 : Topological measurements on the new algorithm.

convex hull. Thus, area of convex hull cannot increase. If area of convex hull becomes zero, all robots must be in a line. Fig.6.9 compares the new algorithm (always succeed) and our previous algorithm [50] (often fail without range-based safety).

Since our new asymmetric algorithm is topology-based, we can also measure total dropping (Fig.6.10(a)), number of reverses (Fig.6.10(b)), number of robots on the main path (Fig.6.10(c)), and maximum sub-tree depth (Fig.6.10(d)). We cannot have these kinds of measurements on our previous geometry-based algorithm.



## Chapter 7

### Conclusion

I implemented distance measurement on the r-one robots, as well as developed a platform for measurement and calibration. Although distance measurement has relatively low resolution and is difficult to improve, it is still sufficient for distributed algorithms.

I presented a straightforward geometry-based algorithm to sort robots in Euclidean space. Robots get sorted from most initial configurations, even if sensor range is limited. However, special cases exist, even after employing range-based safety constrains. There are limited spaces to improve this algorithm.

Then I presented a thoughtful topology-based algorithm to sort robots from arbitrary initial configuration. This algorithm is intrinsically and provably safe, correct, and effective. It is a good solution for large quantity of low-cost robots to sort while moving. My future work is to improve time and distance complexity, and effectively untie knots on the main path of real robots.

## Bibliography

- [1] Y. Litus and R. Vaughan, “Fall in! sorting a group of robots with a continuous controller,” in *2010 Canadian Conference on Computer and Robot Vision (CRV)*, pp. 269–276, May 2010.
- [2] I. Matijevics, “Infrared Sensors Microcontroller Interface System for Mobile Robots,” in *5th International Symposium on Intelligent Systems and Informatics, 2007. SISY 2007*, pp. 177–181, Aug. 2007.
- [3] A. Sabatini, V. Genovese, E. Guglielmelli, A. Mantuano, G. Ratti, and P. Dario, “A low-cost, composite sensor array combining ultrasonic and infrared proximity sensors,” in *1995 IEEE/RSJ International Conference on Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings*, vol. 3, pp. 120–126 vol.3, Aug. 1995.
- [4] M. Garcia and A. Solanas, “Estimation of distance to planar surfaces and type of material with infrared sensors,” in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004*, vol. 1, pp. 745–748 Vol.1, Aug. 2004.
- [5] P. Novotny and N. Ferrier, “Using infrared sensors and the Phong illumination model to measure distances,” in *1999 IEEE International Conference on Robotics and Automation, 1999. Proceedings*, vol. 2, pp. 1644–1649 vol.2, 1999.
- [6] L. Korba, S. Elgazzar, and T. Welch, “Active infrared sensors for mobile robots,”

- IEEE Transactions on Instrumentation and Measurement*, vol. 43, pp. 283–287, Apr. 1994.
- [7] P. Vaz, R. Ferreira, V. Grossmann, and M. Ribeiro, “Docking of a mobile platform based on infrared sensors,” in , *Proceedings of the IEEE International Symposium on Industrial Electronics, 1997. ISIE '97*, vol. 2, pp. 735–740 vol.2, July 1997.
- [8] G. Benet, F. Blanes, J. E. Sim, and P. Prez, “Using infrared sensors for distance measurement in mobile robots,” *Robotics and Autonomous Systems*, vol. 40, pp. 255–266, Sept. 2002.
- [9] A. Schaufelbuechl, U. Munich, C. Menolfi, O. Brand, O. Paul, Q. Huang, and H. Baltes, “256-pixel CMOS-integrated thermoelectric infrared sensor array,” in *The 14th IEEE International Conference on Micro Electro Mechanical Systems, 2001. MEMS 2001*, pp. 200–203, Jan. 2001.
- [10] F. Fang, L. Huang, and M. Zeng, “A low-power low-cost digital readout circuit for dual-band infrared sensor array,” in *7th International Conference on ASIC, 2007. ASICON '07*, pp. 498–501, Oct. 2007.
- [11] V. Pavlov, H. Ruser, and M. Horn, “Model-based object characterization with active infrared sensor array,” in *2007 IEEE Sensors*, pp. 360–363, Oct. 2007.
- [12] Y. Li and C. Ma, “Design and realization of intelligent infrared data communication system,” in *2014 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 1811–1816, Aug. 2014.
- [13] H. Takai, G. Yasuda, and K. Tachibana, “Development of a space-division infrared receiver for fully distributed inter-robot communication networks,” in *2002*

- IEEE International Conference on Systems, Man and Cybernetics*, vol. 5, pp. 6 pp. vol.5–, Oct. 2002.
- [14] F. Arvin, K. Samsudin, and A. Ramli, “A Short-Range Infrared Communication for Swarm Mobile Robots,” in *2009 International Conference on Signal Processing Systems*, pp. 454–458, May 2009.
- [15] J. McLurkin, *Analysis and Implementation of Distributed Algorithms for Multi-Robot Systems*. Ph.D. thesis, Massachusetts Institute of Technology, 2008.
- [16] H. Li, J. McLurkin, and M. Embree, “Physical bubble sort,” tech. rep., Rice University, 2012.
- [17] S. Poduri and G. S. Sukhatme, “Constrained coverage for mobile sensor networks,” in *IEEE International Conference on Robotics and Automation*, (New Orleans, LA), pp. 165–172, May 2004.
- [18] M. Zavlanos and G. Pappas, “Potential Fields for Maintaining Connectivity of Mobile Networks,” *IEEE Transactions on Robotics*, vol. 23, pp. 812–816, Aug. 2007.
- [19] R. Williams and G. Sukhatme, “Locally constrained connectivity control in mobile robot networks,” in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 901–906, May 2013.
- [20] M. Zavlanos, A. Jadbabaie, and G. Pappas, “Flocking while preserving network connectivity,” in *2007 46th IEEE Conference on Decision and Control*, pp. 2919–2924, Dec. 2007.

- [21] M. A. Hsieh, A. Cowley, V. Kumar, and C. J. Taylor, “Maintaining network connectivity and performance in robot teams,” *Journal of Field Robotics*, vol. 25, pp. 111–131, Jan. 2008.
- [22] D. Zhang, L. Mao, Z. Li, and J. Chen, “Infrared communication link maintaining method for multiple mobile microrobots,” in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 2269–2273, Dec. 2013.
- [23] J. Kutylowski and F. Meyer auf der Heide, “Optimal strategies for maintaining a chain of relays between an explorer and a base camp,” *Theoretical Computer Science*, vol. 410, pp. 3391–3405, Aug. 2009.
- [24] B. Degener, B. Kempkes, P. Kling, and F. M. a. d. Heide, “A continuous, local strategy for constructing a short chain of mobile robots,” in *Structural Information and Communication Complexity* (B. Patt-Shamir and T. Ekim, eds.), no. 6058 in *Lecture Notes in Computer Science*, pp. 168–182, Springer Berlin Heidelberg, Jan. 2010.
- [25] M. Dynia, J. Kutylowski, F. Meyer auf der Heide, and J. Schrieb, “Local strategies for maintaining a chain of relay stations between an explorer and a base station,” in *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '07*, (New York, NY, USA), pp. 260–269, ACM, 2007.
- [26] F. M. a. d. Heide and B. Schneider, “Local strategies for connecting stations by small robotic networks,” in *Biologically-Inspired Collaborative Computing* (M. Hinchey, A. Pagnoni, F. J. Rammig, and H. Schmeck, eds.), no. 268 in *IFIP The International Federation for Information Processing*, pp. 95–104, Springer

- US, Jan. 2008.
- [27] T. Balch and R. Arkin, "Behavior-based formation control for multi-robot teams," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, 1998.
- [28] T. Balch and M. Hybinette, "Social potentials for scalable multi-robot formations," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, pp. 73–80 vol.1, 2000.
- [29] M. Egerstedt and X. Hu, "Formation constrained multi-agent control," *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 947–951, Dec. 2001.
- [30] M. Ji and M. Egerstedt, "Distributed Coordination Control of Multiagent Systems While Preserving Connectedness," *IEEE Transactions on Robotics*, vol. 23, pp. 693–703, Aug. 2007.
- [31] W. Ding and Y. Shi, "Preplanned Recovery Schemes Using Multiple Redundant Trees in Complete Graphs," in *Future Control and Automation* (W. Deng, ed.), no. 173 in Lecture Notes in Electrical Engineering, pp. 179–187, Springer Berlin Heidelberg, 2012.
- [32] A. Cornejo, *Local Distributed Algorithms for Multi-Robot Systems*. Ph.D. thesis, Massachusetts Institute of Technology, 2012.
- [33] G. Habibi, L. Schmidt, M. Jellins, and J. McLurkin, "K-redundant trees for safe multi-robot recovery in complex environments," in *ISRR 2013*, 2013.
- [34] J. McLurkin, A. J. Lynch, S. Rixner, T. W. Barr, A. Chou, K. Foster, and S. Billestein, "A low-cost multi-robot system for research, teaching, and outreach," *Proc.*

*of the Tenth Int. Symp. on Distributed Autonomous Robotic Systems DARS-10, October, 2010.*

- [35] J. McLurkin, A. McMullen, N. Robbins, G. Habibi, A. Becker, A. Chou, H. Li, M. John, N. Okeke, J. Rykowski, S. Kim, W. Xie, T. Vaughn, Y. Zhou, J. Shen, N. Chen, Q. Kaseman, L. Langford, J. Hunt, A. Boone, and K. Koch, “A robot system design for low-cost multi-robot manipulation,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)*, pp. 912–918, Sept. 2014.
- [36] OSRAM Opto Semiconductors, *GaAs Infrared Emitter IRL80A*, 08 2007.
- [37] Sharp, *GP1UX31QS Series Holder-less Type IR Detecting Unit for Remote Control*.
- [38] A. Krller, S. P. Fekete, D. Pfisterer, and S. Fischer, “Deterministic Boundary Recognition and Topology Extraction for Large Sensor Networks,” in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, SODA '06*, (Philadelphia, PA, USA), pp. 1000–1009, Society for Industrial and Applied Mathematics, 2006.
- [39] T. Du and Z. Wang, “An improved bearing-only based Semidefinite Programming algorithm for large scale WSN node localization,” in *2014 IEEE 9th Conference on Industrial Electronics and Applications (ICIEA)*, pp. 504–509, June 2014.
- [40] S. Mohammadloo, M. Arbabmir, and H. Asl, “New constrained initialization for bearing-only SLAM,” in *2013 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pp. 95–100, Nov. 2013.

- [41] S. Zhufeng, Z. Yubo, and X. Mingyan, “Bearing-Only TMA Based on Improved Particle Swarm Optimization,” in *Second International Symposium on Computational Intelligence and Design, 2009. ISCID '09*, vol. 1, pp. 108–111, Dec. 2009.
- [42] M. Fardad, S. Ghorashi, and R. Shahbazian, “A novel maximum likelihood based estimator for bearing-only target localization,” in *2014 22nd Iranian Conference on Electrical Engineering (ICEE)*, pp. 1522–1527, May 2014.
- [43] A. Becker, S. P. Fekete, A. Krller, S. K. Lee, J. McLurkin, and C. Schmidt, “Triangulating Unknown Environments Using Robot Swarms,” in *Proceedings of the Twenty-ninth Annual Symposium on Computational Geometry, SoCG '13*, (New York, NY, USA), pp. 345–346, ACM, 2013.
- [44] W. Peterson and D. Brown, “Cyclic Codes for Error Detection,” *Proceedings of the IRE*, vol. 49, pp. 228–235, Jan. 1961.
- [45] Vishay Semiconductors, *Data Formats for IR Remote Control*, 08 2013.
- [46] Philips Semiconductors, *Remote Control System RC-5 Including Command Tables*, 12 1992.
- [47] E. Olson, “AprilTag: A robust and flexible visual fiducial system,” in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3400–3407, May 2011.
- [48] Q. Li and D. Rus, “Navigation protocols in sensor networks,” *ACM Transactions on Sensor Networks*, vol. 1, no. 1, pp. 3–35, 2005.
- [49] S. Madden, R. Szewczyk, M. Franklin, and D. Culler, “Supporting aggregate



queries over ad-hoc wireless sensor networks,” in *Proceedings Fourth IEEE Workshop on Mobile Computing Systems and Applications, 2002*, pp. 49–58, 2002.

- [50] Y. Zhou, H. Li, and J. McLurkin, “Physical bubblesort,” 2014. Poster presented at International Symposium on Distributed Autonomous Robotic Systems (DARS), November 2–5, Daejeon Convention Center, Daejeon, Korea.