The Hong Kong Polytechnic University
Department of Computing

Programming and Coordination in Distributed Multi-Robot Systems

Shan Jiang

16900499R

Supervisor: Prof. Jiannong Cao

A Confirmation Report submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

2017

Abstract

Over the last decades, robotic systems have reached a position of design maturity and are therefore ready for the mass-production of cheap robots. The current trend in robotics community is to use a group of robots to accomplish user-defined tasks instead of using single-robot systems. These group of robots working in collaboration with each other form an ensemble which is commonly referred to as a multi-robot system (MRS). Use of MRS provides better scalability, reliability, flexibility, versatility and helps in performing tasks in a faster and cheaper way as compared to the single-robot systems.

Along with those advantages, there are a lot of challenging issues of managing MRS in both hardware and software parts. First, the large number of robots make the system development extremely complicated. It is tough for a robotic system developer to develop such complex systems that should be robust, scalable, and support the real-time integration of various components. Therefore, a programming model is required to program MRS containing thousands or even millions of robots. Second, it is hard to coordinate a large number of robots in a distributed or decentralized manner. In traditional MRSs, the base stations are used to coordinate the robots. However, this kind of centralized infrastructure makes the system hard to scale up and suffers from single-point failure. These shortages can be overcome by utilizing distributed MRSs. However, coordinating multiple robots in a distributed manner is not trivial. Last but not least, Experimental evaluation and validation are necessary for research in MRS. It often happens that theoretical models and algorithms with perfect simulation results do not work under real-world conditions. Therefore, it is crucial to building a testbed for MRS to conduct multi-robot research.

In this study, we propose a framework for programming and coordination in dis-

tributed MRSs. Based on the framework, we address the aforementioned challenging issues one by one. First, we propose a real-time programming model, namely RMR, to control large-scale multi-robot system. RMR is a logic programming model, which enables programmers to focus on high-level application requirements of "what to do" and leave low-level implementations (e.g., data management and communication) of "how to do" to the runtime system. Second, we develop distributed algorithms to coordinate the multiple robots correctly and efficiently. In particular, we propose a uniform circle formation algorithm, which can be used in the application of area coverage and exploration. Finally, we build a test-bed of distributed MRS. The test-bed includes eight robots and a localization system. We further evaluate the proposed programming model and on our MRS. Also, we develop several impressive demos including "multiple robots pass through narrow corridor" and "multiple robots form different shapes".

In the future, we plan the further investigate the distributed coordination problems in MRSs. To be specific, the context of malicious robots and faulty robots are of our interest.

List of Publications

1. Shan Jiang, Jiannong Cao, Jia Wang, Milos Stojmenovic, Julien Bourgeois, "Uniform Circle Formation by Asynchronous Robots: A Fully-Distributed Approach", accepted by ICCCN 2017.

2. Yuvraj Sahni, Jiannong Cao, Shan Jiang, "Middleware for Multi-Robot System", a chapter to appear in "The Philosophy of Mission-Oriented Wireless Sensor Networks".

3. Shan Jiang, Jiannong Cao, Yang Liu, Jinlin Chen, Xuefeng Liu, "Programming Large-Scale Multi-Robot System with Timing Constraints", ICCCN 2016.

4. Shan Jiang, Junbin Liang, Jiannong Cao, Rui Liu, "An ensemble-level programming model with real-time support for multi-robot systems", PerCom Workshops 2016.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Recent advances in robotics and other related fields have made it feasible for developers to build inexpensive robots. The current trend in robotics community is to use a group of robots to accomplish task objectives instead of using single robot systems. These group of robots working in collaboration with each other form an ensemble which is commonly referred as a multi-robot system (MRS). Use of MRS provides better scalability, reliability, flexibility, versatility and helps in performing any task in a faster and cheaper way as compared to single robot system [2]. MRS system can be very useful in search and surveillance applications especially for areas which are difficult or impossible for humans to access. Another benefit of MRS is that it has better spatial distribution [106]. Many applications such as underwater and space exploration, disaster relief, rescue missions in hazardous environments, military operations, medical surgeries, agriculture, smart home etc. can make use of distributed group of robots working in collaboration with each other [2] [56]. It would not only be difficult but may also lead to wastage of resources if such

applications are developed using single robot systems.

The benefits provided by MRS do not come at low cost. MRS is a dynamic and distributed system where different robots are connected to each other using wireless connection. Robots in MRS should collaborate with each other to perform complex tasks such as navigation, planning, distributed computation, etc. But it is not easy to manage MRS in due to various challenging issues in both hardware and software parts. First, the large number of robots make the system development extremely complicated. It is tough for a robotic system developer to develop such complex systems that should be robust, scalable, and support the real-time integration of various components. Therefore, a programming model is required to program MRS containing thousands or even millions of robots [91]. Second, it is hard to coordinate a large number of robots in a distributed or decentralized manner. In traditional MRSs, the base stations are used to coordinate the robots. However, this kind of centralized infrastructure makes the system hard to scale up and suffers from single-point failure. These shortages can be overcome by utilizing distributed MRSs. However, coordinating multiple robots in a distributed manner is not trivial. Last but not least, Experimental evaluation and validation are necessary for research in MRS. It often happens that theoretical models and algorithms with perfect simulation results do not work under real-world conditions. Therefore, it is crucial to building a testbed for MRS to conduct multi-robot research.

1.2 MRS Applications

Robots contain both sensing and actuator components which makes them useful for a wide range of applications. Applications which involve navigation, exploration, object transport and manipulation benefit from the use of MRS. Researchers have been trying to develop biologically inspired robots that incorporate not only

the structure of insects and animals but also their social characteristics to design multi-robot system. Researchers try to emulate the communication behavior in bees, birds, and other insects to design control and coordination system for MRS. We have classified the robotic applications into seven categories as shown in Fig. 1.1. A brief overview of the robotic application is also provided below. These applications are generic and not specifically related to MRS. However, the current research trend is that most applications are now being developed using MRS instead of single robot system.

- *HealthCare Robots*: Robots have been used by healthcare and medical professionals for a long time. One of the most important uses of robots in healthcare has been for performing and assisting surgeries. Robots are used for performing precise and minimally invasive surgeries [10] [15]. The current research trend in this area is to use biologically inspired robots that can move in confined spaces and manipulate objects in complex environments [15]. Other areas where robots are being used in this application domain is rehabilitation and assistive robotics [43] [101]. Robots are used for recovery of patients with impaired motor and cognitive skills [43]. Robots are being used for assistance to elderly and other physically or mentally disabled individuals to help them live independently. There are even companion robots that help such individuals with special needs. However, due to lack of awareness and other reasons, patients and even healthcare professionals are reluctant to accept robots for medical purpose [101] [12].

- *Industrial robotics*: Robots are now a main component in manufacturing and logistics industries. Industries have been using robots for tasks which are impossible or difficult for humans, such as working in a room filled with hazardous substances, inside a furnace, etc. [43]. Several robotic application

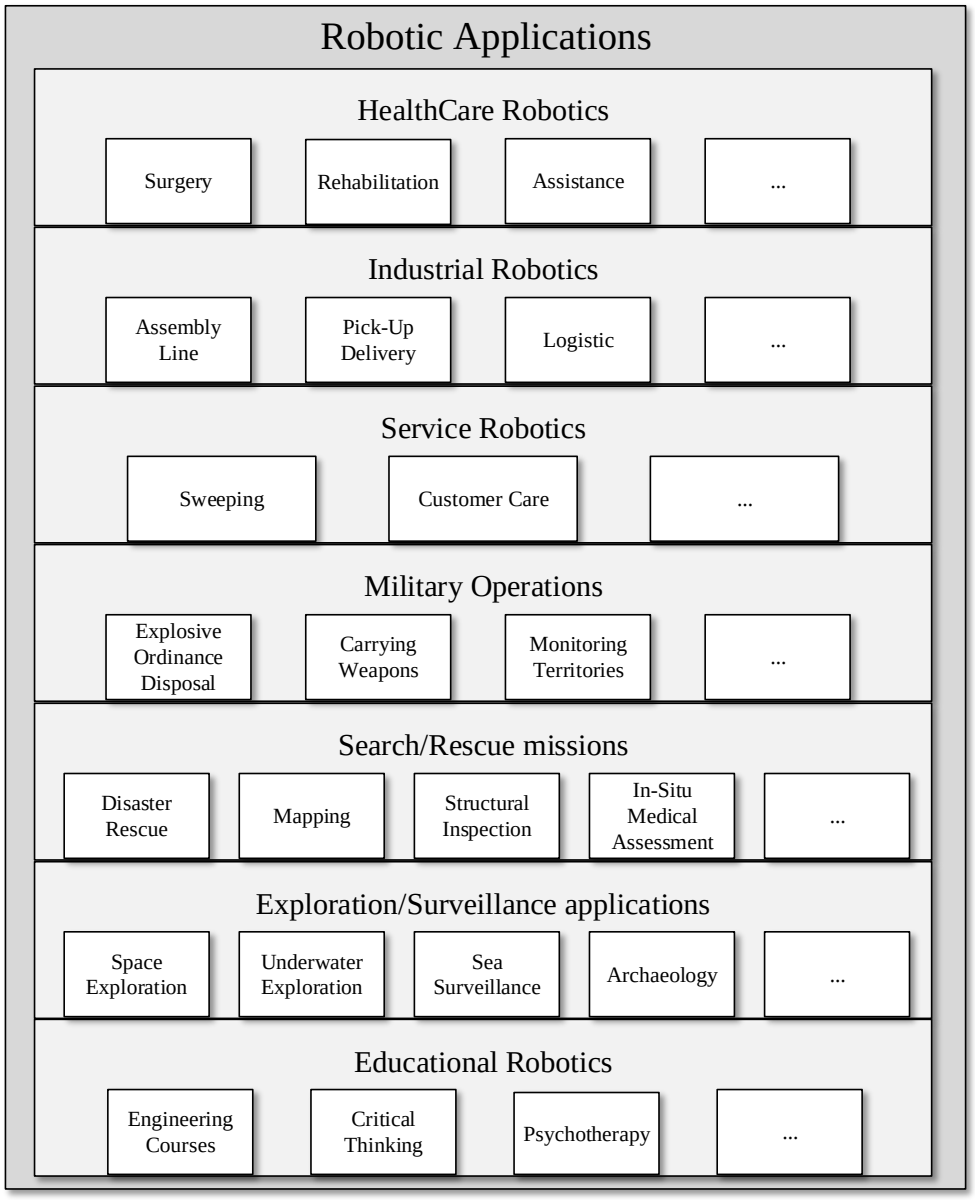


Figure 1.1 Classification of Robotic Applications

studies in manufacturing industries have been mentioned in [33] including die casting applications, forging applications, heat treatment applications, glass manufacturing applications etc. All large-scale manufacturing industries especially automobile, component assembly, and many other industries involving tasks related to packaging, testing, and logistics rely on the use of robots for efficient task completion [94]. Besides automation, robots are also used for assisting humans in their activities in industries.

- *Service robotics*: Service robots are fully or semi-autonomous robots that perform tasks useful for the well-being of humans except in manufacturing related activities. Service robots are useful for performing tasks that are trivial, dangerous, or repetitive for humans. Home service robots are one such type of robots. They can be used for activities that range from cleaning floor, kitchen, bathroom, windows, swimming pool to lawn mowing, washing clothes, and many other activities [43] [94]. Besides home, service robots can also be used for other services such as object pickup and delivery, customer care, etc. [43].

- *Military operations*: Most of the military organizations around the world are using different types of robots for situations that are risky for humans [92]. Robots are also cheaper to maintain than having the human personnel. Military robots can be classified into three categories, which are ground robots, aerial robots, and maritime robots [92]. These military robots are very often used for battlefield surveillance from ground, air and underwater level. Ground robots are also being utilized for explosive ordinance disposal. Besides carrying out surveillance operations in enemy territories, unmanned aerial vehicles (UAVs) are also used for carrying missiles to attack enemy sites.

- *Search and Rescue missions*: Rescue robots are used to provide real-time information about the situation to aid search and rescue missions. Rescue robots are used for performing tasks such as searching in unstructured and hazardous environments, reconnaissance and mapping, rubble removal, structural inspection, in-situ medical assessment and intervention, and providing logical support [94]. Rescue robots can be utilized for many situations including natural disasters, mining accidents, fire accidents, explosions, etc. [43]. Rescue robots are also useful for post-disaster experimentation [94]. A key aspect of this application is that rescue robots must be autonomous and they are supposed to work in an unstructured environment where any pre-existing communication network may not work properly.

- *Exploration/Surveillance application*: Robots are a useful for collecting data in unstructured environments, unknown territories, and from areas which are difficult or impossible for humans to access. Space exploration, underwater exploration, and exploration in hazardous environments such as radiation prone areas, wilderness, mines, damaged buildings, etc. are some examples of this application [94]. Exploration or surveillance is an important part of other applications too such as military operations, and rescue missions. Navigation, coordination, and collaboration are three important tasks performed by robots in surveillance applications. A lot of researchers are trying to develop biologically inspired robots that can navigate in confined spaces and perform complex tasks [53].

- *Educational robotics*: Robots are now being used in schools and universities for the educational purpose also. Students can learn about multiple disciplines such as computer science, electronics, mechatronics etc. by developing robotic applications and learning from the experience [1]. However, there

is a drawback with this approach as students only learn about robot related fields. Several studies have been reviewed in [11] and it is observed that most studies only help in teaching concepts related to physics and mathematics such as Newton's Law of motion, kinematics, fractions, etc. Students who are interested in other fields such as music or arts do not get much benefit out of this. There are few instances where robots have been used for teaching students something different from mathematics or physics. In [104], Lego robots have been used to teach about evolution. Lego robots have also been utilized in [78] to improve social connection in individuals with autism and Asperger's syndrome. This shows that robots have huge potential for contribution towards education. Research efforts are required to find ways to use robots for the development of skills such as critical thinking, problem solving, teamwork, etc.

Chapter 2

DiMRS - a Distributed Intelligent Multi-Robot System

2.1 Introduction

Experimental evaluation and validation are important for research in MRS. It often happens that theoretical models and algorithms with perfect simulation results do not work under real-world conditions. In MRS, these divergences are even more amplified compared with single-robot system due to the large number of robots, interactions between robots, and the effects of asynchronous and distributed control, sensing, actuation, and communication. Therefore, it is crucial to build a testbed for MRS to conduct multi-robot research [71]. In this section, we list key requirements of a MRS and show how robotics community has progressed in building distributed MRS over the years.

One of the earliest multi-robot systems is the Mataric R2 robots built in the 1990s (seen in Fig. 2.1(a)). They use a group of four robots to demonstrate and ver-

(a) Mataric R2 (b) Khepera (c) Khepera II

Figure 2.1 Three kinds of robot for multi-robot system in early age

ify the group behavior such as foraging, flocking and cooperative learning [64]. For each Mataric R2 robot, it equips piezoelectric bump sensors for collision detection, two-pronged forklift for picking goods, six infrared sensors for object detection and radio transceivers for broadcasting up to one byte of data per second. Nearly the same time, the K-Team from Switzerland developed Khepera robot team in 1996 and Khepera II robot team in 1999 [74] seen in Fig. 2.1(b) and Fig. 2.1(c) respectively. The size of the robot is reduced from 36-cm long (Mataric R2 robot) to 8-cm long (Khepera and Khepera II). The Khepera II robot has stronger functionality than the Mataric R2 robot such as more powerful computation ability and more reliable wireless communication. Due to the development of electronic technology, the Khepera II robot also has a smaller size.

After the early age, more and more multi-robot systems are built in both laboratory and industry nowadays. Two representative multi-robot systems are Swarmbot [69] [70] developed by McLurkin and iRobot for research purpose in 2004 (seen in Fig. 2.2(a)) and Kiva [105] developed by Amazon for warehouse usage in 2007 (seen in Fig. 2.2(b)). Also, the research community has organized a lot of multi-robot competitions such as RoboCup for robotic soccer, MAGIC competition for military surveillance and MicroMouse for maze exploration. A lot of multi-robot

(a) Swarmbot (b) Kiva

Figure 2.2 Two representative multi-robot system in recent years: Swarmbot for research purpose and Kiva for industry usage

systems result from these competitions such as AIBO dog [17], NAO humanoid [41] and Cmdragons [13].

We have observed several features of a multi-robot system:

- Cost: inexpensive for each single robot. A general purpose for MRS is to let quantities of agents, each of which owns limited ability, to achieve a complex system-level target. The system must be designed to be inexpensive to allow researchers to incrementally increase the size of the system. When a multi-robot system is scaled up, it will be hard to cover the fee if each individual robot is highly expensive.

- Size: small size for each single robot. Given limited space, robots with the large size may have problems of frequent collisions, communication blocking and less flexibility. Also, robots in huge size go against the scalability of the whole system.

- Functionality: stable and strong sensibility for each single robot. If every robot has stable functionality, the whole system can be reliable enough.

> Stronger the sensibility is, more information it may acquire from itself, the environment and other robots. Hence, the whole system may achieve more complex tasks.

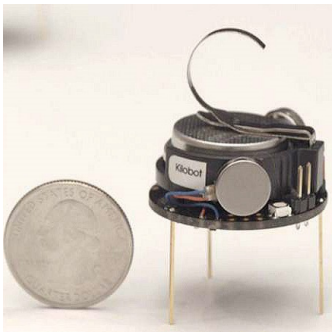
As we know, stronger functionality may result in larger size and higher cost. Therefore, to build a MRS it is crucial to find a balance between cost, size and functionality.

Though there have been a lot of multi-robot systems, most of them are controlled in centralized way. In another word, there is a central controller to schedule the robots to perform cooperative tasks. Centralized multi-robot system can be hardly scaled up due to limited computation capability of the central controller. Hence, scholars transfer their research direction to distributed multi-robot system [34]. There are varieties of active research topics that explore efficient algorithms to control distributed multi-robot system, such as self-reconfiguration [3] [89] and exploration [48] [14]. Scholars generally envision their algorithms to be feasible for a distributed multi-robot system consisting of hundreds, thousands and even more robots [25] [3] [97]. However, these algorithms are usually evaluated in simulator only [3] [89], or deployed on a small group of tens of robots or fewer [51] [49] due to cost, time or complexity. As we previously mentioned, a simulator can hardly model robots' movement, communication and sensibility in a precise way. Therefore, it would be significant if a large-scale distributed MRS can be built up for algorithm evaluation.

A MRS is said to be fully distributed [81] if each robot in the system supports:

- Distributed control: to process gathered information and to make the decision locally while achieving the system-level goal.

- Distributed sensing: to sense itself, the environment and other robots locally.

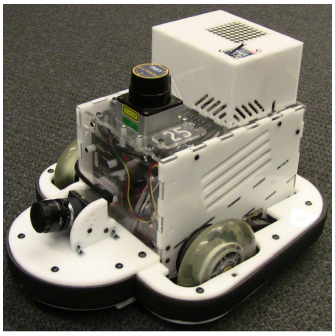


(a) Kilobot

(b) R-one

(c) Khepera IV

(d) E-puck



(e) Scarab

Figure 2.3 Five representative robots suitable for multi-robot system nowadays

- Distributed actuation: to navigate freely in the environment without collision with obstacles and other robots.
- Distributed communication: to receive and transmit data from other robots in a scalable robot network.

2.2 Existing MRS

Table 2.1 A comparison of off-the-shelf multi-robot systems in terms of functionalities and hardware

	Kilobot	R-one	Khepera IV	E-puck	Scarab
Source	Harvard U	Rice U	K-Team	EPFL	Pennsylvania U
Locomotion	vibration	wheel encoders 3-axis gyro 3-axis accelerometer	wheel encoders 3-axis gyro 3-axis accelerometer	wheel encoders 3-axis gyro 3-axis accelerometer	wheel encoders 3-axis gyro 3-axis accelerometer
Sensibility	1 IR range sensor	8 IR range sensors 8 bump sensors 4 light sensors a speaker	8 IR range sensors 8 light sensors 4 IR cliff sensors 5 ultrasonic range sensors 1 microphone 1 speaker 1 color camera	8 IR range sensors 8 light sensors 1 microphone 1 speaker 1 color camera	laser range sensor high-res color camera
Communication	IR signal	IR signal radio	802.11 b/g WiFi Bluetooth 2.0 EDR	radio	radio
Computation	8MHz Atmega328 32kB Memory	50MHz ARM Cortex-M3 64KB SRAM 256KB Flash	800MHz ARM Cortex-A8 512MB RAM 512MB Flash 4GB Flash for data	Microchip dsPIC MCU 8KB RAM 144KB flash Flash	/*
Battery life (h)	3-24	4	7	1-10	/*
Size (cm)	3.3	10	14	7.5	22.2
Cost ($)	14	220	2625	545	3000

* not specified

Knowing basic elements for a distributed MRS, we characterize some typical MRSs in detail and compare their functionality and cost. The criteria are to select open-source, still active and relatively high-impact MRS. The summary of comparison can be seen in Tab. 2.1. In detail, five multi-robot systems are considered as follows: Kilobot [86] [85], r-one [66], Khepera IV [96] (evolved from Khepera III [82]), e-puck [73], Scarab [71].

- The Kilobot[1] (seen in Fig. 2.3(a)) is designed by the K-Team and used in SSR lab of Harvard University. Kilobot is a low-cost robotic system especially suitable for research of swarm robotics. The functionality of each individual Kilobot is limited, i.e., only can sense the distance from its neighbor, sense the intensity of visible light and receive/transfer message from/to its neighbors. However, a collective of Kilobot achieve relatively complicated behaviors such as generating different shapes [88] and transporting large objects [87]. This kind of robotic system in which every robot is with limited ability while can achieve complicated behavior together is called swarm robotics. It is inspired by biological swarm behaviors [80] such as bird flocking and ant manipulation. Another such kind of system is the I-Swarm [52] from the University of Stuttgart. However, the robot Jasmine in I-Swarm is far more expensive ($130) compared with Kilobot ($14) while the functionality is similar. Simple functionality makes low-cost possible, on the other hand, limits the feasible environment. For example, a message is transmitted using the reflection of infrared signals. Therefore, the floor where the Kilobots move must be smooth enough, or infrared signals may not reach individual's neighbors.

[1] http://www.eecs.harvard.edu/ssr/projects/progSA/kilobot.html

- The r-one[2] (seen in Fig. 2.3(b)) is designed and used in Rice University. r-one is a relatively low-cost robot that enables large-scale multi-robot research and education. In terms of locomotion, each robot is equipped with two-wheel encoders, a 3-axis gyro and a 3-axis accelerometer to move on a floor with awareness of odometer, speed and acceleration. With respect to communication, there are two kinds of communication method. First one is to use infrared transmitter and receiver to achieve directional communication and second one is to use radio to achieve nondirectional communication with higher bandwidth. The sensing ability is provided by using 8 bump sensors for 360° detection. r-one provides ample functionalities at a low cost which has motivated its use for education area application [68]. Several courses are taught using r-one . r-one can also be used for multi-robot manipulation [67] and transportation [46] if each robot is equipped with a gripper.

- The Khepera IV[3] (seen in Fig. 2.3(c)) is designed and made by K-Team. It is a commercial robot with abundant and powerful functionality compared with non-commercial ones. A standard Khepera IV has the same equipment for locomotion as r-one. For the communication part, Khepera uses 802.11 b/g Wi-Fi and Bluetooth 2.0 EDR for wireless communication instead of infrared signals or radio. Khepera IV has strong sensibility due to the presence to multiple sensors. A Khepera IV is equipped with five ultrasonic transceivers and eight infrared sensors for obstacle detection, four extra infrared sensors for cliff detection, one microphone and one color camera for multimedia functions and twelve light sensors and three programmable LED for human-robot interaction. Besides, Khepera IV is highly extensible. Developers may extend

[2] http://mrsl.rice.edu/projects/r-one

[3] http://www.k-team.com/khepera-iv

native functions using the generic USB, Bluetooth devices and custom boards plugging into the KB-250 bus. Khepera IV wrap the remarkable abilities of sensing, communication and locomotion in a small body of 14-centimeter diameter. However, the cost of each Khepera IV is over 2600 US dollars. The Khepera series robot is adopted by DISAL of EPFL and is used for various research topics such as multi-robot learning [29] and odor plume tracing [95].

- The e-puck[4] (seen in Fig. 2.3(d)) is designed and made by EPFL. E-puck designer Francesco Mondada started with the Khepera group and moved to make simpler education robots. An e-puck is equipped with two-wheel encoders, a VGA camera, three omnidirectional microphones, 3-axis accelerometer, eight infrared sensors and eight ambient light sensors. Also, e-puck is only 7cm long and easy to extend functionality. For instance, rotating scanner and turret with three linear cameras are two optional extensions. E-puck is specially designed and widely used for education purpose [20]. It is used in the teaching areas of signal processing, automatic control, behavior-based robotics, distributed intelligent systems and position estimation and path finding of a mobile robot [73]. In addition, e-puck is also used in many research topics such as supervisory control theory [60] and distributed control strategy [93].

- The Scarab shown in Fig. 2.3(e) is designed and made at the University of Pennsylvania. Compared with other robots, the design of Scarab shifts from minimal multi-robots to a complex and robust system. Two of the major components in a Scarab is the Hokuyo URG laser range finder and the Point Grey Firefly IEEE 1394 camera. Using the laser and camera, Scarab is capable of the tasks requiring strong sensibility and high computation payload such as

[4] http://www.e-puck.org/

SLAM (simultaneous localization and mapping) [84] and vision processing. However, A Scarab is significantly large, heavy and expensive with 23cm diameter, 8kg weight and over 3000-dollar cost. Consequently, Scarab is not practical for large populations i.e., more than ten Scarabs working together. But using less than five Scarabs for multi-robot SLAM is applicable [90].

Chapter 3

Programming Large-Scale Multi-Robot System with Timing Constraints

3.1 Introduction

The remarkable progress of robotics technology has made it feasible to deploy a large number of inexpensive robots with complicated tasks. The robots together form a multi-robot system (MRS), which has better reliability, flexibility, scalability and versatility than a single-robot system. There has been quantities of applications for MRS, such as multi-robot exploration, multi-robot surveillance and multi-robot manipulation. However, the management of the robots is a challenging issue. Among the difficulties towards managing MRS, one of the most important is the lack of dedicated tools. In particular, one problem that is significant but has received little attention is the programmability. To be specific, a scalable program-

ming model is required to program MRS containing thousands or even millions of robots. Moreover, the programming model is supposed to support setting timing constraints on the behaviors of the robots, which is required in many real-time multi-robot applications.

Traditionally, robots are programmed using an imperative programming paradigm. Such programming tasks are expensive in terms of time consumption and code complexity. For example, the robots in our lab are programmed with embedded C, which is an imperative programming language. There are some other domain-specific imperative programming models such as TinyOS [47], Swarm [72], Paintable Computing [16], and CAs [63], all of which focus on the behavior of individual devices instead of the aggregate.

General multi-robot applications require coordinated movements with real-time decision making capabilities in an unstructured environment. Hence, with respect to programming a group of robots, the tasks of communication and coordination needs to be abstracted instead of being explicitly written for each of the robots. There have been several research efforts to develop such programming models. Unfortunately, all of them have focused on a dedicated programming model for a specific application.

Early success in the development of programming models that enable programmers to think on a macroscale arose from the field of overlay networks and sensor networks. P2[59] and SNLog [19] showed that logic programming approach could be used to allow an ensemble to be programmed as a whole. Other such programming models for sensor networks include Hood [103], TinyDB [62] and Regiment [75]. However, these sensor network programming models are limited by their focus on sensing and data gathering without attending to actuation and control. Moreover, they presume a static network of immobile nodes which changes infrequently due to node failures. A notable exception is Pleiades [55], which could be used

in situations with dynamic network topologies of sensor networks. But owing to adoption of a programming style similar to OpenMP [23], it eventually leads the programmers to focus on individual modules instead of the whole ensemble.

Inspired by the logic programming approach for sensor networks, researchers tried to extend their application for mobile robots. For example, Proto [9], which is effective for programming stationary sensor-actuator networks as a whole, has been extended to mobile robots as Protoswarm [7]. Protoswarm is a functional language that uses *Amorphous Medium Abstraction* [6] for programming MRS. LDP [26] is derived from a method for distributed debugging but is originally designed for modular robotics. While it works well in highly dynamic systems, it can lead to excessive messaging in more static environments.

Meld [4][5] is another logic programming language for modular robots that enables the programmer to specify high-level logic of what is to be decided or achieved, and leaves the low-level details of data manipulation and communication to the implementation of the programming language. However, it is restricted to applications on modular robots which are independently executing modules robots where inter-robot communication is limited to immediate neighbors. Moreover, for many real-time multi-robot applications, besides being logically correct, satisfying certain temporal constraints is a hard demand to successfully achieve final targets with high precision. Meld doesn't provide any mechanism to support real-time scheduling of the robots.

In this work, we proposed and designed a new programming model for MRS, called RMR. It follows the logic programming paradigm, which enables it to achieve high scalability. Moreover, it allows developers to specify timing constraints on the behaviors of the robots, such as setting deadlines and identifying the time orders of actions. To support distributed execution of RMR programs, a compiler and a runtime system are developed for RMR. The compiler is able to convert the RMR

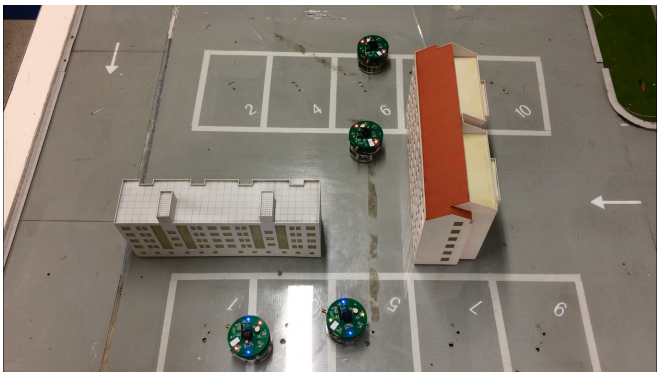


Figure 3.1 Example application: multiple robots are passing through a narrow corridor

programs into executable byte-codes, and then distribute the byte-codes to each robot. The runtime system is responsible for interpreting and executing the byte-codes. To evaluate the performance of RMR, we deployed RMR in a simulator and a realistic test-bed, and then developed several example applications. Fig. 3.1 shows one of the applications utilizing RMR, in which multiple robots cooperate with each other to pass through a narrow corridor. Our main contributions are:

- We designed and proposed a new programming model called RMR for MRS. RMR allows programmers to specify timing constraints for large-scale MRS in a easy-to-use fashion.

- We implemented and deployed RMR in both a simulator and a realistic test-bed. To support distributed execution of RMR programs, we developed a compiler and a runtime system. Furthermore, we did solid real-world experiments to evaluate the performance of RMR.

The reminder of the paper is structured as follows. Section 3.2 introduces the design philosophy and main features of RMR. In Section 3.4, we first describe the compiler of RMR, and then present the mechanisms in our runtime system to support the distributed execution of RMR programs. The deployment of RMR in

both simulator and realistic test-bed and the evaluation in introduced in Section 3.5. Section 3.6 concludes the paper with some future improvement directions.

3.2 Design Principle

Logic programming has a long history and there exist a number of variants of logic programming languages. Commonly, a logic programming language encompasses a set of facts and rules as its basic elements. Facts can be used to specify system states, sensed physical events, system configuration, etc, while rules allow programmers to describe how the system evolves. In practical applications, it is important to design good abstractions to mask the complexity of programming MRS, and meanwhile provide real-time guarantee for the coordination of the MRS. To this end, the following principles are considered on designing RMR.

3.2.1 Ensemble-level abstraction

With respect to a group of robots assigned with a task, it is nature to think about what the robot ensemble as a whole should do. This leads us to consider the deign principle of ensemble-level abstraction. The entire MRS can be viewed as a single and monolithic unit while programmers write RMR program. RMR enables a developer to think about what the whole set of robots should do in a global and easy-to-understand perspective, and hide the detailed and complex implementation of how to do. This greatly simplifies the programming process, and benefits the developers to program MRS with a large number of robots. This abstraction technique is also used in programming modular robots [4], and is referred to as macroprogramming in the area of wireless sensor networks [44].

3.2.2 Combination of forward and backward reasoning

In logic programming, forward reasoning and backward reasoning are two main methods of reasoning. Backward reasoning starts with a list of goals and works backwards from the consequent to the premises to see if the consequent is available. Backward reasoning is often used to specify the direction of reasoning, e.g. in Prolog [76]. One the contrary, forward reasoning starts with the available premises and uses rules to derive new facts until the goal is reached. Forward reasoning is often used to speed up the execution of programs, e.g., in Meld [22].

Both backward reasoning and forward reasoning have advantages and disadvantages. It is natural to expect that the general performance of the system could be improved by combining the two kinds of reasoning. RMR successfully combines forward and backward reasoning in the execution of RMR programs. In this way, RMR will derive facts with specific direction while guarantee the execution speed of the RMR programs.

3.2.3 Declarative specification of timing constraints

In MRS, it is likely that a series of reasoning steps are involved in order to perform a job. When a developer writing a program, he mainly concern about the global deadline of finishing the entire job rather than the local deadlines of individual reasoning steps. For example, in wireless sensor-actuator networks, the system need to respond in real-time to the physical world events captured by sensors. Actually, this is a complicated "sensing-decision-actuation" process involving many intermediate steps, such as data aggregation for complex event detection. It is highly desirable that the developers only need to describe the global deadline of the entire job and need not to worry about how the local deadlines can be meet at the intermediate steps. This style is referred to as a declarative specification of timing constraints,

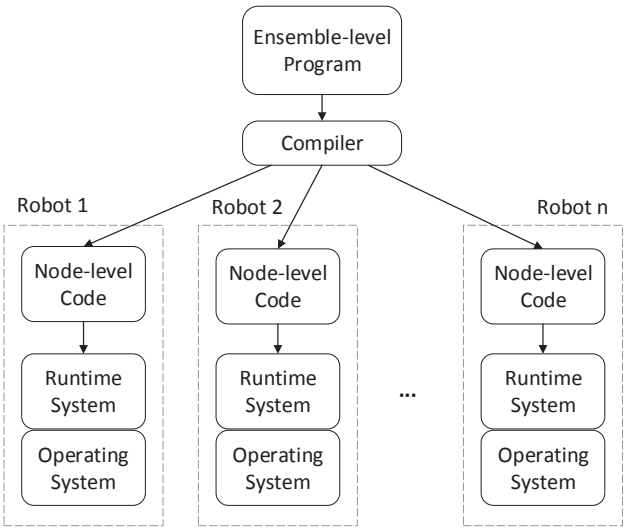


Figure 3.2 Overview of RMR Compilation

which has been incorporated in RMR.

With the above principles, RMR language will create a new way of writing programs for MRS. An overview of programming steps is shown in Fig. 3.2. The highest level is the RMR program, which is written by programmers and obeys a centralized, ensemble-level abstraction. A RMR compiler in the middle level is able to convert the ensemble-level program into node-level byte-codes ran on individual physical robots. To support the execution of such node-level code, a runtime system embedded in the operating system is needed. The runtime system plays the key role in ensuring the efficiency of distributed reasoning and the satisfaction of deadline requirements. In the following sections of this paper, we will introduce these components one by one.

3.3 Language Specification

The design principles in Section 3.2 enable us to design a language that can greatly simplify programmers' thinking processes and reduce their development efforts. In the following, we will describe the detailed specification of RMR. RMR comprises the following main elements.

3.3.1 Variable, Constant and Boolean Expression

RMR follows the conventions of logic programming model when defining variables and constants. Variable names begin with an uppercase letter, whereas constant names begin with a lowercase letter. Moreover, RMR supports boolean expressions that can be calculated into boolean values (true or false).

3.3.2 Fact

Facts are generally in the form of predicates, and they can return boolean values according to the results whether the facts are satisfied or not. Similar to constants, the identifier of a fact should begin with a lowercase letter. A fact generally has a predicate symbol (or name) and can take some arguments (variables) as input. The first argument of a fact must be a robot, which indicates the owner of the fact. Typically, a fact can be used to represent a system state, a detected event, or a relation between its arguments. In RMR, facts are divided into three categories: persistent fact, temporary fact and goal fact.

- Persistent facts refers to those that hold permanently, such as ID of a robot. They can not be consumed along the program lifetime, and must be declared with a bang mark '!', which means "of course".

```
// initial fact that R is in s
location(R, s).
// goal fact that go to middle if possible
?location(R, middle).

// rule 1: if R is in s, then go to middle
location(R, L), L = s -o location(R, middle).
// rule 2: if R is in s, then go to t
location(R, L), L = s -o location(R, t).
// rule 3: if R is in middle, then go to t
location(R, L), L = middle -o location(R, t).
```

Figure 3.3 RMR program to move a robot from area `s` to area `t` passing through area `middle` without *action*

- Temporary facts are those representing a temporary state and can be consumed. For example, `movealong(A,L)` is a temporary fact, which means an intermediate state that robot `A` is moving along the line `L`.

- Goal facts represent the goals of the program that must be satisfied if possible. They must be declared with a question mark '?', which means "why not" or "to be achieved".

Persistent fact and temporary fact are generally used in logic programming language while goal fact is proposed in our programming language and is one of the new elements. Goal facts enable programmers to specify intermediate states in their codes. This function is not supported in other logic programming language.

For example, if we want a group of robots to go from area `s` to a specified area `t` while passing by area `middle` if possible. In traditional logic programming language for robotic system, such as Meld [4], we can write that there is an initial fact that the robots are in area `s`. Furthermore, there are three rules. The first one is

that if the robots are in area `s`, then they can go to area `middle`. The second one is that if the robots are in area `s`, then they can go to area `t`. The last one is that if the robots are in area `middle`, they can go to area `t`. Since there are two areas `middle` and `t` to which the robots can go from `s`, the robots will randomly choose one of them to go to. This is true because we can not specify the procedure how the robots go from `s` to `t` in traditional logic programming language.

However, we can achieve it by adding a new element called goal fact into logic programming language. The program is shown in Fig 3.3. In the program, we first write down the initial fact and the three rules. Furthermore, we add a goal fact `?location(R, m)`, which means that "Why not go to the area middle if possible". In this way, when the program runs, it will only choose the the first rule to apply and go to location `middle` if possible.

3.3.3 Action

As we have seen in Section 3.3.2, there are nothing involved with actuation and motion in the robotics system. To support the motion control for realistic robots in language, we introduce the concept of *action*. An action also has a predicate symbol and can take some arguments as input. The first argument of an action also must be a robot, which is the executor of the action. Unlike fact, actions must be declared as an action type in previous, so that the actions will not stored and will be consumed immediately. The introduction of "action" connects the programming language and the realistic robot. A fact will only influence the execution of the program, while an action can have effect on the physical environment and the robot itself. To be specific, when an action is derived, there will be some underlying function call according to the action rather than being inserted in to the database of the facts. The programmers are able to define actions on their own.

```
type action moveto(robot, area).

location(R, s).
?location(R, middle).

// use action moveto instead
location(R, L), L = s -o moveto(R, middle).
location(R, L), L = s -o moveto(R, t).
location(R, L), L = middle -o moveto (R, t).
```

Figure 3.4 RMR program to move a robot from area `s` to area `t` passing through area `middle` with *action*

```
function moveto(Area p) {
    // control the robot to go to Area p
    ...
    // insert the location fact back
    VM.insert_fact(type_location, p)
}
```

Figure 3.5 The function *moveto* written in the operating system of a robot

We add actions in the example mentioned in Section 3.3.2 and the modified program is shown in Fig. 3.4. We will explain how the robotics system evolves in the following. At the head of the program, a predicate of `moveto` is declared as a new action type. Firstly, there is a initial fact `location(R, s)` meaning that the robot is in the area `s`. Then the runtime system will apply the first rule, delete the fact `location(R, s)`, and derive an action `moveto(R, middle)`. The action `moveto(R, middle)` will not be inserted into the database of the facts but will call the underlying function `moveto` in the operating system, which is demonstrated in Fig. 3.5. The function `moveto` will control the robot to go to the area `middle` and insert a new fact `location(R, middle)` back into the runtime system of the robot. In this time, the robot will have a new fact `location(R, middle)` and the robotics system continues to evolve.

3.3.4 Rule

Rules have the following structure:

$$p_1, p_2, \cdots, p_m \text{ -o } q_1, q_2, \cdots, q_n$$

where $q_i (i = 1, 2, \cdots, n)$ are facts or actions, and $p_i (i = 1, 2, \cdots, m)$ are facts or boolean expressions. The interpretation of the above expression is that all q_is can be derived if all p_is in the body of the rule are satisfied. Commas in the body are interpreted as logical conjunction. The rules make the derivation of new facts and new actions from existing facts possible.

For example, a rule

```
online(A,L) -o {B | edge(A,B) | ready(B,A)}
```

means that if robot `A` is on the line `L`, then a fact that `A` is `ready` will be derived in any robot `B` who has an `edge` with `A`. Here, the braces are the symbol of a compre-

hension, which enables the repeated application of a rule and is also used in other logic programming language. Also, the programs in Fig. 3.3 and Fig. 3.4 also contain some examples about the usage of rules.

3.3.5 Time assertion

We develop a new construct in RMR called time assertion to allow developers to specify timing constraints in declarative fashion. Specifically, the time assertion for a real-time job A should be described in the following form:

$$assert(s_A, f_A, d_A, v_A)$$

In the time assertion, s_A and f_A are facts (predicates) referring to the system states when the real-time job A starts and ends, respectively; d_A is the deadline that needs to finish the job A; v_A is an action that will be derived if a violation of the time assertion is detected. Let us denote the physical time when the system state enters s_A and f_A by t_A and t'_A, respectively. The above time assertion requires $t'_A - t_A \leq d_A$, or v_A will be derived.

For example, a time assertion

```
assert(offline(A,L), online(A,L), 10, broadcast(A))
```

means that if robot `A` does not move to the line `L` within `10` seconds since the last time it is not on line `L`, it will `broadcast` a failure message.

3.4 Compiler and Runtime System

In this section, we will first introduce the function of our compiler, and then provide a suit of runtime system to support the distributed execution of RMR programs.

3.4.1 Compiler

The compiler is responsible for translating the code written in RMR to byte-codes. The byte-codes will be easier to be interpreted by the robots than the RMR code. Inside the byte-code file, there are hexadecimal data about the number of the facts, the number of the rules, the offset to the description of the first fact, the offset to the description of the fist rule and so on. Since the computational capability of each robot is limited, it will be much better to read such hexadecimal data than the strings in the original RMR code. During the compilation of RMR programs, the ensemble-level code will be transferred to node-level code. Then the node-level code will be distributed to each robot for interpretation and execution.

3.4.2 Workflow of the Runtime System

To run the byte-codes generated by the RMR compiler, we developed a runtime system. The workflow of the runtime system is shown in Fig. 3.6.

When the robot is started, the runtime system will also be started using the function `VM.startVM`. Inside the function, the runtime system will be initialized with the byte-codes at first. The initialization includes initializing the database of the facts, inserting basic facts into the database, interpreting and storing the rules, handling the actions and the time assertions, etc. Then, the runtime system will see if the state of the system can be updated using the function `VM.computePredicate`.

In the function `VM.computePredicate`, the runtime system will see if there is any satisfied rule. If so, a piece of selected rule will be applied, which means facts may be deleted or added and actions may be derived. Then, if there is any new actions or new facts derived due to the applied rule, there will be events passing from the runtime system to the operating system of the robots. For example, if new actions are derived, there will be an event called `NEW_ACTION` received in the op-

```
void Robot.receiveEvent(EveType type, ArgList list)
    if type is NEW_ACTION
        processAction(n, list)
    if type is NEW_FACT
        Wait Until VM is not busy
        VM.computePredicate()

Rule VM.selectOneRule()
    highPriority = list()
    lowPriority = list()
    for i in range(0, NUM_RULE)
        if rule(i) is satisfied
            if goal facts can be derived in rule(i)
                highPriority.push(rule(i))
            else
                lowPriority.push(rule(i))
    if highPriority is not empty
        return a random element in highPriority
    if lowPriority is not empty
        return a random element in lowPriority
    return null

void VM.computePredicate()
    busy = true
    rule = selectOneRule()
    if (rule is null)
        busy = false
        return
    Process The Rule rule
    if there is any Action ac derived:
        robot.receiveEvent(NEW_ACTION, ac)
    if there is any Fact derived:
        robot.receiveEvent(NEW_FACT)
    busy = false

void VM.startVM()
    Do Initialization With The Byte-codes
    VM.computePredicate()
```

Figure 3.6 The Workflow of the Runtime System

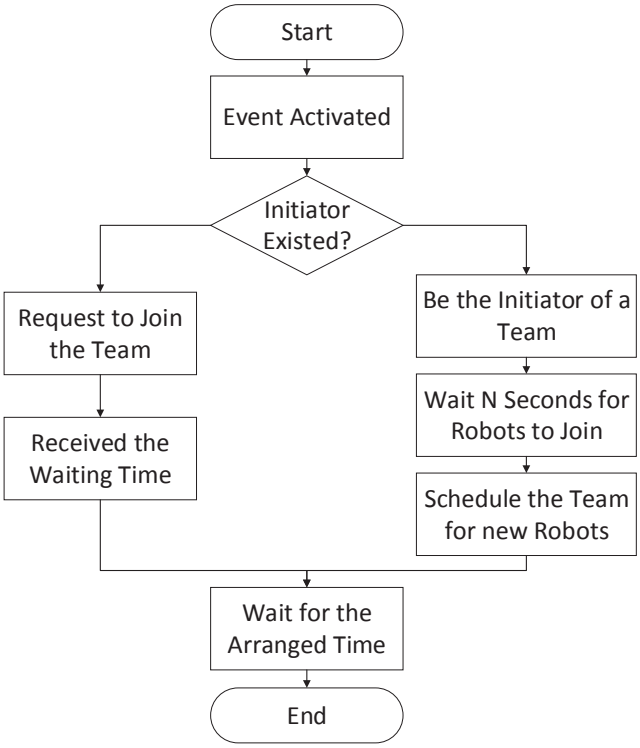


Figure 3.7 Distributed Scheduling

erating system of the robot, and the robot will react accordingly using the function `Robot.receiveEvent`.

Now, we come back to the function `VM.selectOneRule`. This function is used to select a piece of satisfied rule if any. To be specific, if there are any satisfied rules, this function will classify the rules into two categories on the basis of whether goal facts can be derived. The rules in which goal facts can be derived will have higher priority to be applied. If there are multiple rules with same priority, the function will randomly choose one to return. The usage of different priorities of different rules makes the functionality of the goal facts possible.

3.4.3 Distributed Scheduling

In the previous part 3.4.2, we figured how the facts, rules and actions work in the runtime system. In this part, we will figure how the time assertions work. In Section 3.3.5, we introduce the specification of time assertion. In a time assertion, if the starting event s_A happens, the time assertion will be triggered. The robot will be aiming at its finish event f_A, which can be seen as its target. When multiple robots are aiming at their individual targets, there may be high resource competition of time and space among the robots. For example, when two robots are going to pass through a narrow corridor at the same time, collision may happen if there is no cooperation and coordination between them. To address such issue, we proposed and implemented a distributed scheduling algorithm in the runtime system. Fig. 3.7 shows the flow chart of our algorithm. To be specific, our distributed scheduling algorithm will schedule multiple robots with different deadlines for a same event to make more robots satisfy their deadlines.

For a single robot `A` in which an event `E` is triggered, it will broadcast a message to see if there has already been a leader for the event `E` at first. If there is a piece of response message that robot `B` is the leader for event `E`, then robot `A` will transmit a piece of message to robot `B` for purpose of joining the group of event `E`. Otherwise, robot `A` will create a new group and serves as the initiator and leader of the event `E`. After that, robot `A` will wait n seconds for new participators of event `E`. The waiting time n will be properly set according to the deadline of event `E` in robot `A`. If robot `A` waits for too long time, robot `A` will be more likely to miss the deadline. If the waiting time is too short, the size of each group may be too small, which leads to little coordination and worse performance. For a leader of an event `E`, it will update the scheduling list if there is a robot trying to join the group of event `E`. The scheduling list is maintained according to the deadline for the event.

3.4.4 Path Planning

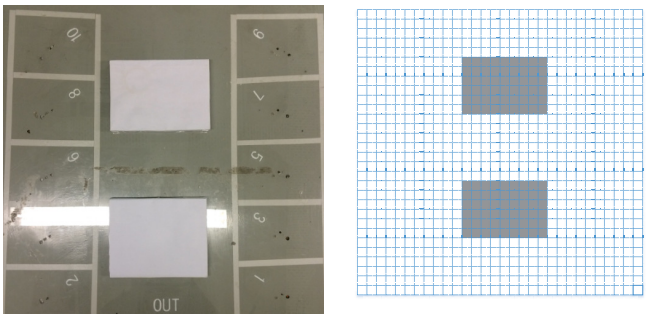
When a robot is scheduled to start moving, it has to plan its path from its current position to its destination. In this phase, we will introduce the path planning problem for a group of robots. In our test-bed, we found that the physical distances between robots in the same group are not too large. In the path planning algorithm, we can utilize such property to save energy. Therefore, we will address the path planning problem separately for the first robot to move (leader) and other robots (followers).

Different Algorithms for the Leader and the Followers

With respect to the leader, we utilize grid-based A* search algorithm as the path planning algorithm, which is demonstrated in the following four steps:

- Overlay a grid graph on the working space
- Map the starting position S_1, the target position T_1 and all the obstacles O_1, $O_2, \cdots, O_m$ into grid points S'_1, T'_1, $\{O_{11}, O_{12}, \cdots, O_{1n_1}\}$, $\{O_{21}, O_{22}, \cdots, O_{2n_2}\}$, $\cdots$, $\{O_{m1}, O_{m2}, \cdots, O_{mn_m}\}$.
- Utilize A* search algorithm to find a shortest path P'_1 from S'_1 to T'_1. The result P'_1 will be reduced based on the grid.
- Output the path planning result P_1 as $S_1 \mapsto S'_1 \mapsto P'_1 \mapsto T'_1 \mapsto T_1$

For the followers, they will make full use of the leader's planning result to determine their paths. The planning result of robot k will be $P_k : S_k \mapsto S'_1 \mapsto P'_1 \mapsto T'_1 \mapsto T_k$. In this way, we can save the energy, especially when the working space is considerably huge.



(a) Realistic Environment (b) Grid Overlay

Figure 3.8 Grid Overlay for the Environment

Grid Overlay

The size of our test-bed is approximately $1.5m \times 1.5m$, which acts as the working space. We overlay a $N \times N$ grid graph on the working space. The decision of N is of significance. If N is too large, the cost of path planning will be too huge. Otherwise if N is too small, the obstacles in the working space cannot be properly abstracted. In our work, we assign N to be 30, by which the test-bed is divided into nine hundred $5cm \times 5cm$ small grids. Fig. 3.8 shows the realistic environment of our test-bed and the corresponding grid graph in which N equals 30. In the abstract grid graph, the grey grids represents the obstacles while the white grids stands for the reachable places.

Grid-Based A* Search Algorithm

After overlaying a grid graph on the working place, we want to find a path from the starting position `S` to the target position `T` in the grid graph. First, we define the connectivity in the grid graph: two grids are connected if their differences in x-axis and y-axis are no more than 1. Moreover, we define the distance between two connected grids as the Euclidean distance. That's to say, the distance between

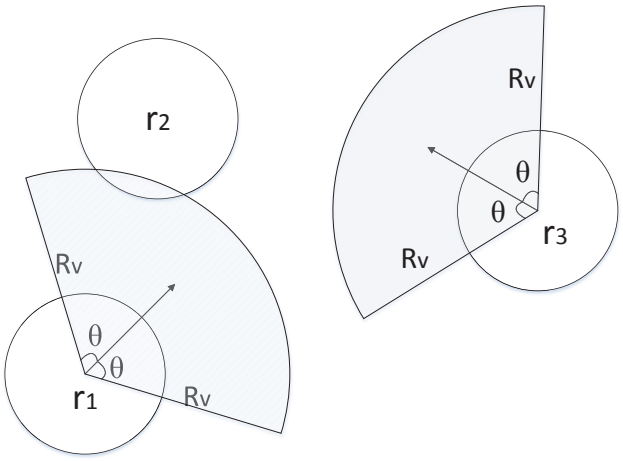


Figure 3.9 the Field of View of the Robots

two grids sharing an edge is 1 while the distance between two connected grids which do not share an edge is $\sqrt{2}$. Then, we use a best-first search strategy to find a least-cost path from S to T. As we traverse the grid graph, we build up a tree of partial paths. The leaves of this tree are stores in a priority queue that orders the leaf nodes according to a cost function, which combines a heuristic estimate of the cost to each T and the distance traveled from S. In detail, the cost function is $f(n) = g(n) + h(n)$. Here, $g(n)$ is the known cost of getting from S to node n, which is tracked by the algorithm, and $h(n)$ is a heuristic estimate of the cost to get from n to T. In our algorithm, $h(n)$ is set as the Euclidean distance between node n and T. We can prove that $h(n)$ is admissible since $h(n)$ never overestimates the actual cost to go to T.

3.4.5 Collision Avoidance

When the path of each robot has been determined, the robots will start moving in the working space. At this point, collisions may happen between robots from different

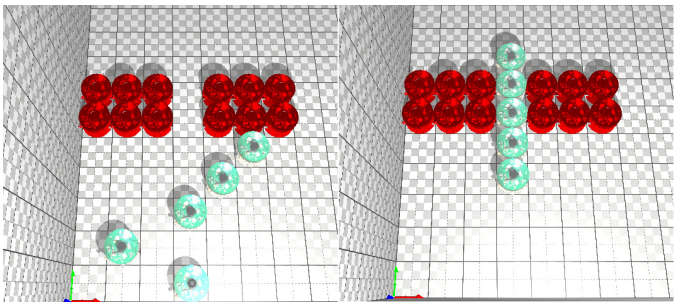


Figure 3.10 Simulation: multiple robots pass through a corridor

groups. Furthermore, in the same group, there can also be collisions on account of out-of-control. Therefore, efficient mechanism is supposed to be proposed to guarantee collision-free movements. To achieve collision-free movements, we divide the mechanism into two aspects: collision detection and collision avoidance.

With respect to collision detection, we can define the field of view by three parameters: orientation, angle of view and depth of view. Then problem of collision detection is solved with knowledge of computational geometry. To be specific, if there is a robot R_2 who enters the field of view of another robot R_1, we say collision is detected in robot R_1, which is shown in Fig. 3.9. Our strategy of collision avoidance is based on a principle that "stop if dangerous". If some others robots or obstacles appear in the field of view of a robot, it will stop its motion at once, and then turn to another direction to continue its movement.

3.5 Deployment

To demonstrate the usefulness and evaluate the performance of RMR, we deployed RMR in a simulator and a realistic test-bed, developed two example applications, and tested the execution time of the first example application.

3.5.1 Simulation

Our simulator is adapted from VisibleSim[28]. The simulator is even driven, which means that everything is modeled as an event and is scheduled for processing. The robot in the simulator maintains a list of events ordered by time to be processed. It always consumes the one at the top of the event list. Consuming an event means calling its associated callback function. For the application developers, there are only two steps to do to add a user-defined event. The first step is to create a new type of event inherited from the base event and the second step is to override the callback function of processing it.

In VisibleSim, only wire communication is supported. We have altered the communication approach from wire communication to wireless communication by implementing a message pool. The message that a robot want to send will be pushed into the message pool. After that, the message pool will deliver the message to the target robots. The robots who receives a piece of message will receive an event called `ReceiveMessage`. The robots are able to process the message by processing the event. Furthermore, features of wireless communication, such as package loss rate, bandwidth, communication range can also be simulated.

When our simulator starts to work, it first use RMR compiler to compile RMR programs into byte-codes. Then, it initializes and renders the working space according to a configuration file. Finally, its virtual machine interprets and runs the byte-codes. Fig. 3.10 shows the simulation that six robots pass through a narrow corridor. In the simulation, the six robots first form into a line facing the corridor and then pass through the corridor in order.

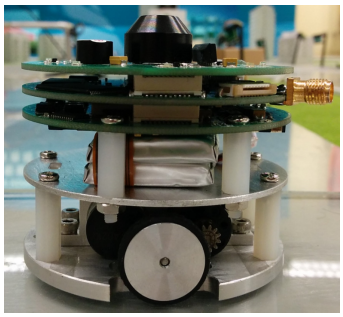


Figure 3.11 Real Picture of a Robot

3.5.2 Real-world Experiments

Our realistic test-bed is composed of three components, which are a localization system, multiple (currently 9) intelligent robots and a programming environment.

Localization System

The location system consists of two ultrasonic sensor. Utilizing ultrasonic sensors which broadcast ten times per second, each intelligent robot is able to get its position on the domo platform.

Intelligent Robots

Our test-bed contains a set of robots, which use FreeRTOS [8] as their operating system.

The robots are home-made in our laboratory. The real picture of one robot is shown in Fig. 3.11. All the robots are in the same shape and with the same size, which is approximately a cylinder with 7cm radius and 18cm height. At the bottom of the robots are the wheels and motors. On the top of each robot, various sensors are equipped. In the middle part, there is a 8.4V lithium battery, which supply power for the whole robot.

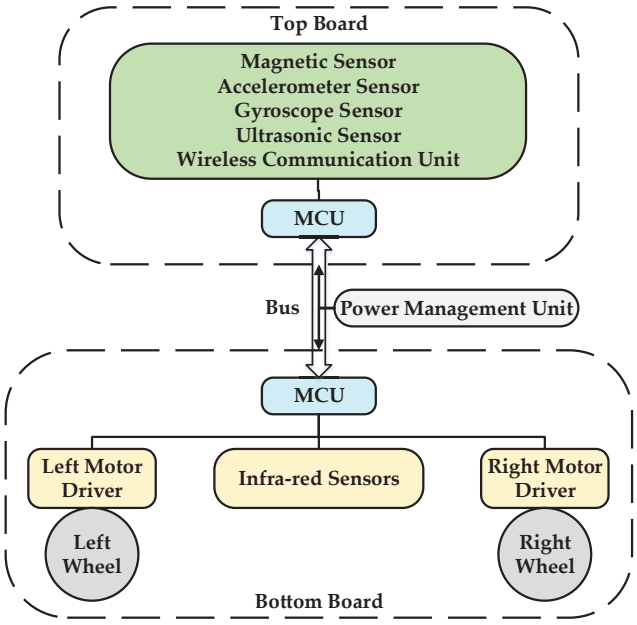


Figure 3.12 Structure Diagram of a Robot

The robot's structure diagram is shown in Fig. 3.12. Each robot is powered by the power management unit. In each robot, the microcontroller unit (MCU) is responsible for data storage and processing. Inside the MCU, data can be stored in either 8Mbit static random access memory (SRAM) or 512Kbit flash memory. Each robot is driven by the motors driver unit, which can control the left motor and right motor separately. The wireless communication unit, which uses IEEE 802.15.4 as its protocol, enables communication between robots. Received signal strength indicator (RSSI) can be used to evaluate distances between it and other robots. The sensors unit is used to acquire information from external environment, which contains an accelerometer sensor, a gyroscope sensor, an ultrasonic sensor and a magnetic sensor.

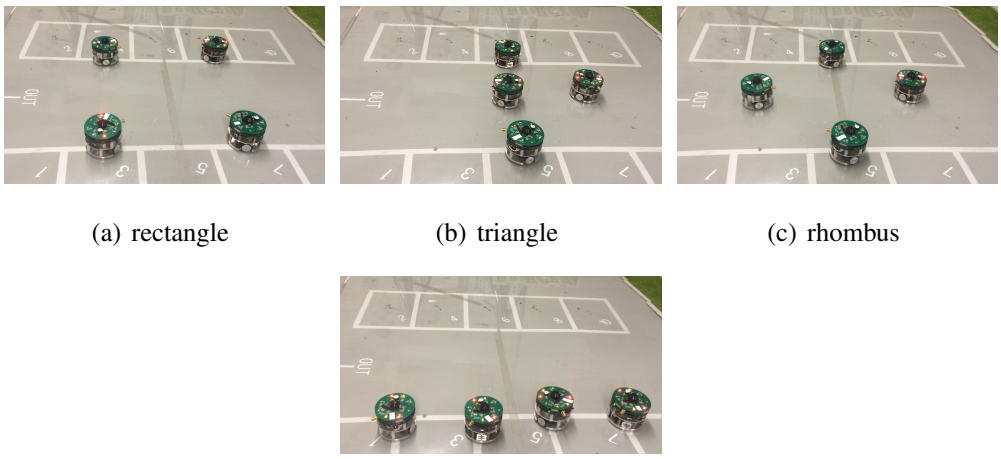
Equipped with different kind of sensors, the robots have various functions. Also, more sensors can be equipped on the robots if necessary. For example, the ac-

celerometer sensor can be used to calculate the moving distance. The magnetic sensor is used to determine the orientation of each robot. And the ultrasonic sensor, communicated with the beacons in the localization system, the robots can be aware of where they are on the platform.

Programming Environment

The programming environment is based on FreeRTOS[8], a popular real-time operating system kernel for embedded devices. In FreeRTOS, programmers can define a set of tasks with priority to be executed concurrently. For example, a task to control motors, a task to get information from embedded sensors, a task for localization and a task for purpose of user-defined application. Then, in every time unit (1/60 second in this test-bed), the tasks will be executed one by one according to the predefined priorities. If not all tasks can be finished during the time unit, tasks with lower priorities may be neglected.

Based on FreeRTOS, we successfully deployed RMR in our robots with three steps. At first, we compile the RMR source program into node-level byte-codes using the compiler and write them into the SRAM of each robot. Then, we create a task as the runtime system for each robot to interpret and run the byte-codes. Finally, we write the callback functions in the robots' operating system triggered by the runtime system if any. To summary, RMR is deployed into the robots by creating a parallel task as the runtime system for RMR program. In addition, a tiny database is implemented to manage (insert, delete, and query) the facts and the time assertions are stored in a priority queue (sorted by time) and are triggered by hardware interrupts.



(a) rectangle (b) triangle (c) rhombus

(d) line

Figure 3.13 Example application: formation control of multiple robots

3.5.3 Example Applications

We developed two example applications to demonstrate the usefulness of RMR. Fig. 3.1 and Fig. 3.13 show demos implemented by RMR in our test-bed. In Fig. 3.1, multiple robots are attempting to pass through a narrow corridor. The robots coordinate with each other to form a line formation facing the corridor, and then move through the corridor in order. In Fig. 3.13, formation control is implemented to enable four robots to generate different shapes of formations. The robots first form into a rectangle in Fig. 3.13(a). Then they form into the shape of triangle in Fig. 3.13(b), rhombus in Fig. 3.13(c), and line in Fig. 3.13(d) respectively.

We further evaluate the efficiency of the runtime system in the first demo. In detail, we conduct experiments in our test bed to observe how execution time changes as the number of robots increase. Here, the execution time refers to the total time spent on starting up robots, moving forward based on the planned path and lining up at the destination. We run the first application with from one to five robots and count the execution time. As illustrated in Fig. 3.14, the result shows that the total execu-

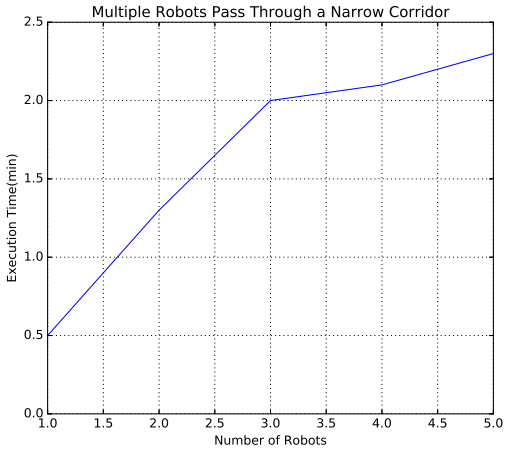


Figure 3.14 Execution time of system with the increasing number of robots

tion time increase slowly as the number of robots increase, which demonstrates the efficiency of our runtime system.

3.6 Conclusion

While programming MRS with real-time requirements is a difficult task at present, it can be greatly simplified by making use of appropriate programming models. In this paper, we presented RMR, a new programming model targeting at programming large-scale MRS with timing constraints. With the new elements "action" and "time assertion" proposed in RMR, RMR enables programmers to specify motions and real-time requirements in multi-robot tasks. After designing RMR, we developed a compile system and a runtime system to support the distributed execution of RMR programs. Furthermore, we have deployed RMR in a simulator and a test-bed to demonstrate the usefulness and evaluate the performance of RMR. By means of experiments in a real deployment, we claimed that RMR is easy-to-use programming model for multi-robot applications with timing constraints. In the

future, we can improve the performance of the runtime system by proposing more efficient mechanisms of distributed scheduling and others. Also, we can improve the programming model to enhance the real-time support for MRS.

Chapter 4

Uniform Circle Formation by Asynchronous Robots: A Fully-Distributed Approach

4.1 Introduction

In recent years, advances in robotics, microelectronics and other related fields have made it feasible for engineers to fabricate inexpensive robots. It has been a trend in the robotics community to use a set of robots to accomplish the tasks instead of a single robot. The group of robots working in collaboration with each other is commonly referred to as a multi-robot system (MRS) [61][79]. The use of MRS provides better scalability, reliability, flexibility, versatility and helps in performing the tasks in a faster and cheaper way compared to single robot systems [2]. MRS can be very useful in search and surveillance applications, in particular in areas which are difficult or impossible for humans to access. Another benefit of MRS is that

they have better spatial distribution [106]. Many applications such as underwater and space exploration, disaster relief, rescue missions in hazardous environments, military operations, medical surgeries, agriculture and smart homes can make use of distributed group of robots. It would not only be difficult but also may result in wastage of resources if such applications are developed using single robot systems.

In many multi-robot applications, a group of autonomous robots is required to eventually form a predefined geometric pattern such as a circle [18][27] or a line. This problem, namely *pattern formation problem* [40], is one of the most important coordination problems for MRS. There are various advantages to forming a pattern such as enhancing coordination efficiency of the system and reducing outer impacts on the system. The pattern formation problem is also closely related to the agreement problem, which is a fundamental problem in distributed computing. For example, forming a single point corresponds to the *gathering* problem requiring all robots to gather at the same location, not determined in advance.

Pattern formation problem has been a hot research topic in the field of MRS for a long time [77]. A particular pattern extensively studied in literature is the *uniform circle* [30][31][36][37], in which the points form a regular polygon. The corresponding problem is called *uniform circle formation*. The problem of uniform circle formation plays a major role in the coordination problems in MRS due to the critical observation of formability by Suzuki and Yamashita [100]. Their observations indicate that uniform circles and points are the only patterns formable from arbitrary initial configuration in FSYNC (and thus also in SSYNC and ASYNC) [37].

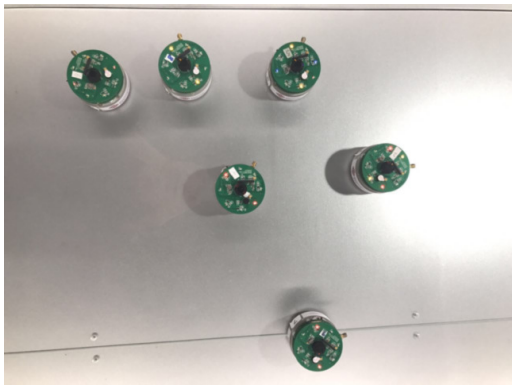
The problem of uniform circle formation can be easily solved in a centralized MRS, in which there is a central computing station. The station knows all the information of the system and is responsible for computing all the actions of the robots. In this case, the problem of uniform circle formation can be mapped as a *minimum weight maximum matching problem (MWMMP)*, in which a set of robots

are assigned with a set of target formation positions, and the weight is the sum of distances from each robot to its goal position. MWMMP is a typical combinatorial optimization problem, which can be modeled as an integer linear programming problem and optimally solved in polynomial time [21].

However, the uniform circle formation problem becomes very challenging for distributed MRS, in which there is no centralized computing station. The robots have to make decisions by themselves independently, e.g., when and where to move, and how to avoid collisions. Moreover, each robot can only communicate with other robots inside its communication range. In this case, the robots do not even know the total number of robots. Therefore, there are many difficulties on how to make a consensus on the circle to form and how to achieve the uniform circle through distributed coordination among the robots.

In this paper, we consider the problem of uniform circle formation for distributed MRS [50]. To solve the problem, we first decompose it into three parts, namely consensus on circle, circle formation and uniform transformation. In the part of consensus on the circle, the robots estimate the total number of robots in the MRS and decide the center and the radius of the circle to form. In the part of circle formation and uniform transformation, the robots form a circle first and then move along the circle to make themselves evenly distributed on the circle. After designing our algorithm, we deployed it in our test-bed. Fig. 4.1 shows the running examples in which six, seven and eight robots are forming uniform circles respectively. We can see that our algorithm can successfully arrange the MRS into uniform circles with different numbers of robots and different initial configurations. The contributions of this work are:

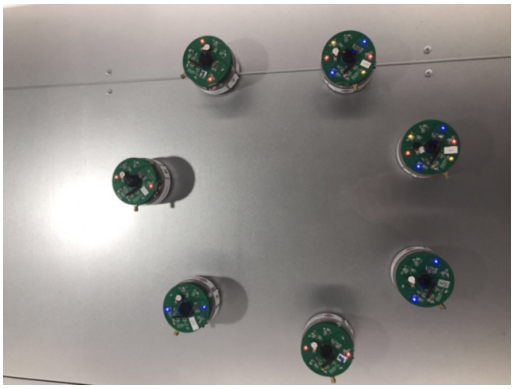
- We formulate the uniform circle formation for distributed MRS and propose a three-phase solution, namely consensus on the circle, circle formation and



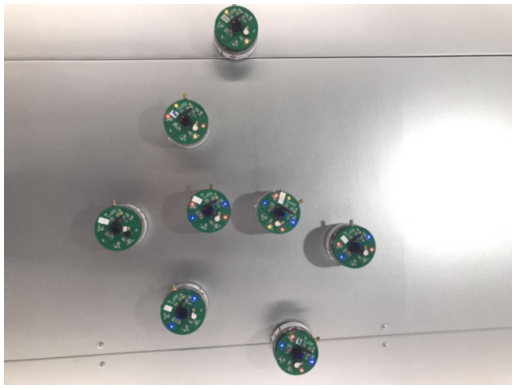
(a) 6 robots, initial configuration

(b) 6 robots, final configuration

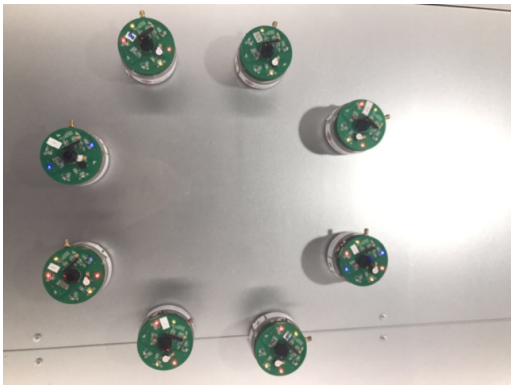
(c) 7 robots, initial configuration



(d) 7 robots, final configuration



(e) 8 robots, initial configuration



(f) 8 robots, final configuration

Figure 4.1 several robots are forming uniform circles

uniform transformation, to this problem.

- We propose distributed algorithms for convex hull construction and cardinality estimation for MRS. We evaluate their performance and efficiency by simulation and theoretical analysis. The results show that our algorithms outperform existing ones. Furthermore, these algorithms can widely be applied in other distributed systems.

- We deploy in a realistic test-bed to test our solution. Successful experiments have implied the practicability and effectiveness of our solution.

We organize the rest of the paper as follows. Section 4.2 introduces the system model and problem definition. A three-phase fully-distributed approach, as well as the simulation and theoretical analysis, are discussed in Section 4.3. In Section 4.4, successful deployment in our test-bed is demonstrated. Finally, related works are discussed in Section 4.5 and Section 4.6 concludes the whole paper.

4.2 Preliminaries

In this section, we introduce the computational model of the system in Section 4.2.1 and formally formulate the uniform circle formation problem in Section 4.2.2.

4.2.1 System Model

Consider a set of n computational entities $\mathcal{R} = \{r_1, \cdots, r_n\}$, namely *robots*, located on a Euclidean plane $\mathbb{R}^2$, on which they can move continuously. The robots are capable of localization, communication, and sensing. For robot r_i at time t, it is aware of its position and orientation $p_i^t = (x_i^t, y_i^t, \theta_i^t)$ under a common Cartesian coordinate system, where θ_i^t is the clockwise angle to the direction of the y-axis. Each

robot can send and receive messages to and from its neighbors within a common communication range R_c. Besides communication, each robot can detect the relative positions of its neighbors within a common sensing range R_s. All robots are of identical size R, which is the radius of the smallest circle that wraps the robot. The only way to distinguish different robots is to use their unique identifiers $r_1, \cdots, r_n$. The identifiers can only be conveyed via wireless communication.

Definition (Configuration) $C^t = \{p_1^t, \cdots, p_n^t\}$ is the collection of positions of all robots at time t.

Definition (Collision-free Configuration) A configuration C^t is said to be collision-free if the Euclidean distance between p_i^t and p_j^t is no less than $2R$ for all $1 < i < j < n$.

Definition (Active Range) The active range R_a of the robots is defined as the minimal value between communication range R_c and sensing range R_s. Two robots are connected if they are within the connected range of each other.

Definition (Connected Configuration) A configuration is said to be connected if for all p_i^t, there exists at least one p_j^t ($j \neq i$) such that the Euclidean distance between p_i^t and p_j^t is no more than R_a.

In the beginning, we assume that the whole system $\mathcal{R}$ is in a collision-free and connected configuration. This means the robots do not collide with each other, and there is no isolated robot in the system. Note that the assumption of full connectivity is necessary, otherwise the whole system will be divided into several small groups, and the problem is going to be solved in several small MRSs separately. Fig. 4.2 gives an example of an initial configuration of 8 robots. In the example, each robot can only sense and communicate locally, and all robots are identical in the system and form a distributed MRS.

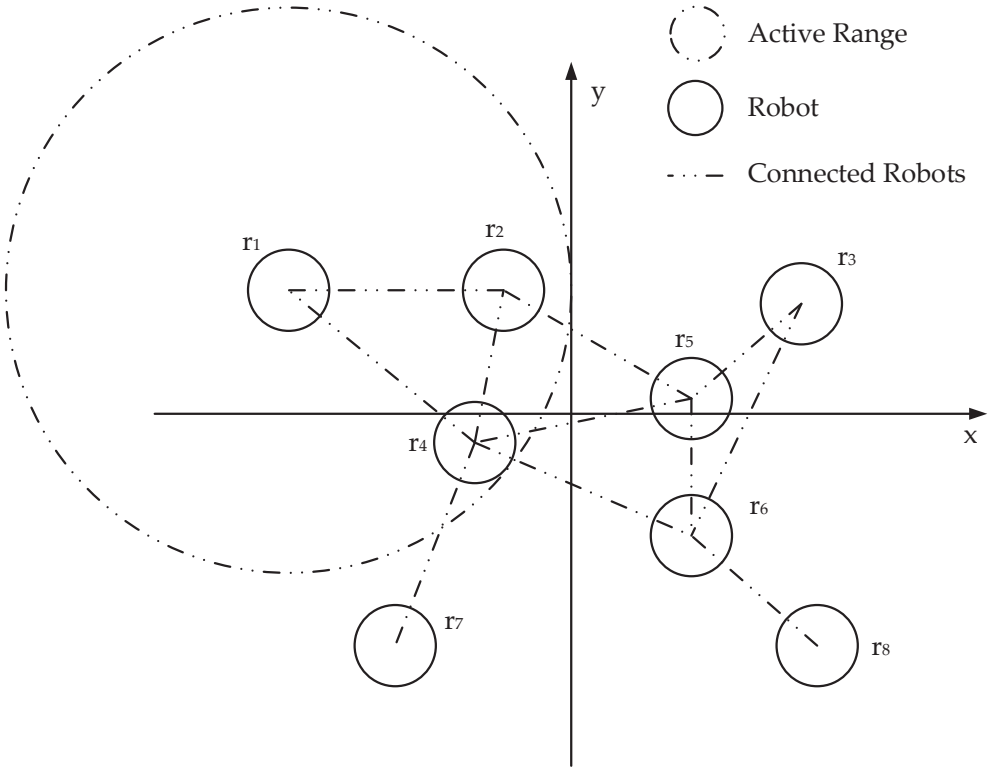


Figure 4.2 An example of initial configuration. It is *collision-free* and *connected*.

Then, the robots repeatedly execute *Sense-Process-Act* cycles. Each cycle can be divided into three sequential phases as follows:

- *Sense*. The robot observes and collects data in the environment, itself and its neighbors. The collected data includes the position of itself, the relative positions of the robots within R_s and the messages from the robots within R_c;
- *Process*. The robot executes a given deterministic algorithm, which takes the collected data as input and outputs the next position to go to and the messages to deliver. Note that the algorithm is the same for all robots;
- *Act*: The robot moves directly toward the destination point (or stays still)

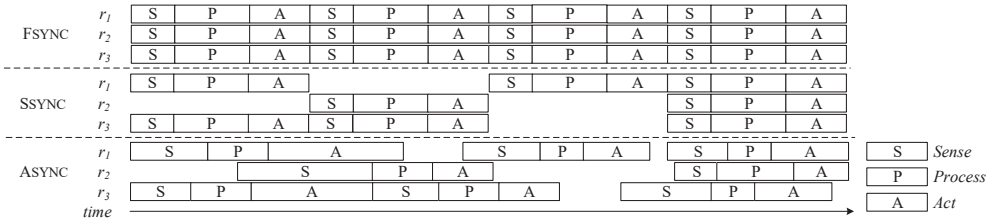


Figure 4.3 An illustration of execution of the *Sense-Process-Act* cycles of three robots for the models of FSYNC, SSYNC, and ASYNC.

along a line segment and delivers the messages. The destination point and the messages to deliver are computed in the previous *Process* phase.

The robots are asynchronous with respect to their *Sense-Process-Act* cycles. That is to say, the execution of each *Sense-Process-Act* cycle of each robot is not only completely arbitrary but independent of the cycles of the other robots as well. In particular, it can be an arbitrarily long duration from the time a robot collects data to the time it moves based on the data. Also, one robot may be starting the *Sense* phase while another robot is performing the *Act* phase. This computational model is called asynchronous (ASYNC, also called CORDA) [38], which is the most general model in distributed systems.

There are two other computational models FSYNC and SSYNC, which have more restrictions than ASYNC. The robots are fully synchronous (FSYNC) if all robots start every *Sense-Process-Act* cycle simultaneously and synchronously execute each of its *Sense*, *Process*, and *Act*. In the semi-synchronous (SSYNC, also called SYM or ATOM) [99] setting, not all robots are active in every cycle, but all of those who start a certain cycle synchronously execute each of its *Sense*, *Process*, and *Act*. Fig. 4.3 compares the three models of execution of the *Sense-Process-Act* cycles.

Definition (Uniformly-circular Configuration) A configuration C^t is said to be uniformly-

circular if there exists a regular n-gon P_n such that each p_i^t $(1 \leq i \leq n)$ equals the position of one of P_n's vertex.

4.2.2 Problem Definition

With the definitions given in the Section 4.2.1, we now formally define the problem of uniform circle formation to be solved in this paper as follows.

Definition Uniform Circle Formation: Given a set of n robots $\mathcal{R} = \{r_1, r_2, \cdots, r_n\}$ initially under a collision-free and connected configuration, arrange them under a collision-free and uniformly-circular configuration through a sequence of collision-free configurations.

4.3 A Fully-Distributed Approach

In this section, we propose a fully-distributed approach to solving the uniform circle formation problem stated in Section 4.2.2. In Section 4.3.1, a general framework of the algorithm is introduced. Then detailed steps are discussed from Section 4.3.2 to Section 4.3.7.

4.3.1 Algorithm Framework

When the system is just starting up, the robots are not aware of their neighbors. A robot network should be built up to enable the communication between the robots. This step is called *network construction*. In Section 4.3.2, a network construction algorithm is proposed to construct neighbor lists for the robots. Note that this step is critical since the topology of the network remarkably affects the number and size of messages passing among the robots.

After the network construction, the robots can communicate with their neighbors. Since the circle to be formed is not given in advance, the robots should negotiate with each other to make a consensus on a common uniform circle to form. Two parameters, the radius, and the center are necessary to determine a uniform circle.

On the one hand, to make a consensus on the center, we propose a distributed convex hull construction algorithm in Section 4.3.3. After execution of the algorithm, each robot will be aware of the convex hull of all robots. Then each robot makes the average of the positions of all robots as the center.

On the other hand, to reach consensus on the radius, a distributed cardinality estimation algorithm is proposed in Section 4.3.4. In the MRS, no robot is aware of the total number of robots. Also, to count the exact number of entities inside a distributed system is computationally expensive. Therefore, our method is to estimate the approximate number of robots in the system. Note that the estimation is crucial since overestimation leads to unconnected configuration while underestimation results in insufficient space to place all robots.

Definition (Circular Configuration) A configuration C^t is said to be circular on center point O and radius r if the distances between each p_i^t $(1 \leq i \leq n)$ and O are equal to r. Furthermore, a configuration C^t is said to be C-circular if C is a circle and C^t is circular on the center and radius of C.

Definition Circle Formation: Given a circle C and a set of n robots $\mathcal{R} = \{r_1, r_2, \cdots, r_n\}$ initially under collision-free and connected configuration, arrange them under a collision-free, connected and C-circular configuration through a sequence of collision-free configurations.

Definition Uniform Transformation: Given a set of n robots $\mathcal{R} = \{r_1, r_2, \cdots, r_n\}$ initially under a collision-free, connected and C-circular configuration, arrange them

under a collision-free and uniformly-circular configuration through a sequence of collision-free configurations.

After reaching a consensus on the common circle, we decompose the uniform circle formation problem into two sub-problems, namely circle formation and uniform transformation seen above. Informally speaking, the two steps are to form a circle first and then to transform into a uniform circle. Here, we assume for simplification that all robots will not isolate themselves from the system during the movements of the whole system.

4.3.2 Network Construction

In this section, each robot constructs a local communication network with identifiers by running Algorithm 1. Initially, each robot r_i broadcasts a message containing its identifier and its position p_i to the neighboring robots within R_c. Upon receiving a piece of message $Msg_j = \{r_j, p_j\}$, robot r_i will testify whether there is any other robot within the circle whose diameter is the segment with ending points p_i and p_j. If there is no other robot in between, r_i will add r_j into its neighboring list N_i.

By running Algorithm 1, each robot r_i can construct a new local communication network N_i, which is also called the 1-hop neighbors of robot r_i. Compared to the original network which is a circle with radius R_c, the number of messages can be remarkably reduced by running the algorithm of convex hull construction and cardinality estimation. It is obvious that all the communication routes in the whole network are bi-directional.

To demonstrate the advantages of utilizing our algorithm of network construction, we compare it with the original network by running the convex hull construction algorithm. We run the distributed convex hull construction algorithm in our network and the original network in an MRS with 3 to 1000 robots and 10 different

Algorithm 1 Network construction for each robot r_i

Input: r_i: robot identifier; R_a: active range; $S_i^{relative}$: the relative positions of the robots within R_s

Output: N_i: neighbor list of robot r_i

Begin:

1: $N_i \leftarrow \emptyset$ ▷ Initialize the neighbor list as an empty set
2: $S_i^{absolute} \leftarrow S_i^{relative} + (x_i, y_i)$
3: Broadcast the message containing the identifier and the position $Msg_i = \{r_i, p_i\}$ to its neighbors
4: **while** Receive a message $Msg_j = \{r_j, p_j\}$ from a neighboring robot **do**
5: $d_{ij} \leftarrow distance(p_i, p_j)$ ▷ Measure the distance between r_i and r_j
6: **if** $d_{ij} < R_a$ **then**
7: $circle_{ij} \leftarrow$ a circle with midpoint of r_i and r_j as the center, and d_{ij} as the diameter
8: **if** No points in $S_i^{absolute}$ are inside $circle_{ij}$ **then**
9: $N_i \leftarrow N_i \cup \{r_j\}$
10: **end if**
11: **end if**
12: **end while**
13: **return** N_i

End

initial configurations for each number of robots. Then, we calculate the average number of messages needed in our network and the original network. Fig. 4.4(a) shows the results, in which `UDG` (unit disk graph) means the original network while `DT` (Delaunay triangulation) means our network. It is oblivious that our network can significantly reduce the number of messages.

4.3.3 Convex Hull Construction

In this section, we propose a distributed convex hull construction algorithm (seen in Algorithm 2) to make all of the robots aware of the convex hull of all robots.

In the algorithm, each robot r_i maintains a local convex hull $conv_i$. When the algorithm is starting up, the primary convex hull for each robot r_i only contains the position p_i itself. Then, each robot r_i exchanges its local convex hull with all of its 1-hop neighbors. Upon receiving a convex hull from another robot, the robot locally runs Graham's scan [42] to merge its local convex hull with the received one. The resulting convex hull is saved as the new local convex hull. Each robot continues this procedure until the local convex hulls in two successive rounds are identical.

Algorithm 2 Distributed convex hull Construction for each robot r_i

Input: N_i: One hop neighbor list; p_i: position; r_i: robot identifier

Output: $conv_i$: the convex hull of the MRS

Begin:

1: **if** not initialized **then**

2: $conv_i \leftarrow \{(r_i, p_i)\}$ ▷ Each robot maintain a local convex hull as $conv_i$. Initially, the convex hull of each robot is only the position of itself.

3: $conv_i' \leftarrow conv_i$

4: $Fin \leftarrow \emptyset$ ▷ Fin records the robots whose messages have been handled in current round.

5: **for** each $r_k \in N_i$ **do**

6: $msglist(r_k) \leftarrow$ an empty message queue

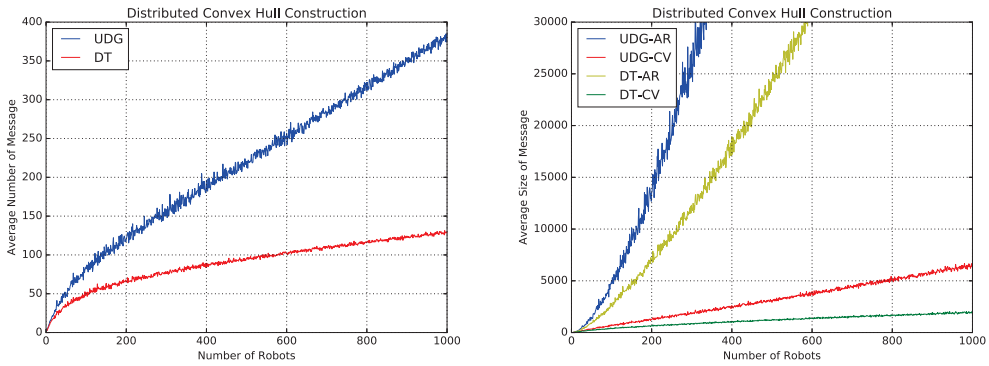
7: **end for**

8: Broadcast $Msg_i = \{r_i, conv_i\}$ to its neighbors

9: **end if**

10: **if** receive message $Msg_j = \{r_j, conv_j\}$ **then**
11: **if** $r_j \in Fin$ **then**
12: $msglist(r_j).push(conv_j)$
13: **else**
14: $conv'_i \leftarrow merge(conv'_i, conv_j)$ ▷ Merge two convex hulls by running Graham's scan algorithm locally
15: $Fin \leftarrow Fin \cup \{r_j\}$
16: **if** $|Fin| = |N_i|$ **then**
17: **if** $conv'_i = conv_i$ **then**
18: **return** $conv_i$
19: **end if**
20: $conv_i \leftarrow conv'_i$
21: broadcast $conv_i$ to r_i's neighbors
22: $Fin \leftarrow \emptyset$
23: **for** each $r_k \in N_i$ **do**
24: $conv_k \leftarrow msglist(r_k).pop()$
25: $conv'_i \leftarrow merge(conv_i, conv_k)$
26: $Fin \leftarrow Fin \cup \{r_k\}$
27: **end for**
28: **end if**
29: **end if**
30: **end if**
End

Another straightforward algorithm to make all robots aware of the common convex hull is that each robot sends all of its neighbors to its 1-hop neighbors repeatedly. In this algorithm, every robot can know all the robots in the system finally



(a) Average number of messages (b) Average size of messages

Figure 4.4 Analysis on number of message and size of messages

and can compute the convex hull. To evaluate our distributed convex hull construction algorithm, we compare it with the straightforward algorithm. We run the distributed convex hull construction algorithm and straightforward algorithm in `DT` network and `UDG` network in an MRS with 3 to 1000 robots and 10 different initial configurations for each number of robots. Then, we calculate the average number of messages as well as the average size of messages needed. Fig. 4.4 shows the results, in which `AR` means the straightforward algorithm while `CV` means our algorithm. According to the figure, we can see that the number and size of messages can be significantly reduced by using our convex hull construction algorithm.

4.3.4 Distributed Cardinality Estimation

In this section, we propose Algorithm 3 to estimate the number of robots in the whole system. Algorithm 3 is adapted from Algorithm 2 by modifying the content of messages and adjusting the updating rule upon receiving messages. After the execution of Algorithm 3, all robots will have a common sense on x_a^i for all $a \in [1, k]$. These parameters will be used for estimating the total number of robots in

Section 4.3.5.

Algorithm 3 Distributed cardinality estimation for each robot r_i

Input: N_i: One hop neighbor list; l: an integer parameter; k: another integer parameter

Output: $\hat{n}_i$: the estimation of total number of robots

Begin:

1: **for** $a \leftarrow 1$ to k **do**

2: $x_a^i \leftarrow$ a variable of l bits with all bits to be 0

3: $rand \leftarrow$ a random 0/1 string of $l-1$ bits

4: $y \leftarrow$ the number of leading continuous '0's in $rand$ from left hand

5: Set x_a^i's $y+1$ bit to be 1

6: $x_a'^i \leftarrow x_a^i$

7: **end for**

8: In Algorithm 2, incorporate $\{x_1^i, \cdots, x_k^i\}$ into message Msg_i. The updating rule of x_a^i is $x_a^i \leftarrow (x_a^i \mid x_a^j)$ when receiving Msg_j. Finally, when returning $conv_i$, also returns x_a^i for all $a \in [1, k]$

End

The general idea of Algorithm 3 is that each robot selects one of l slots to be placed and finally makes a consensus on all the occupied slots. This procedure is repeated k times. In the algorithm, l and k are parameters which can be used to achieve different accuracy requirements. The details will be discussed in Section 4.3.5.

4.3.5 Consensus on Circle

In this section, each robot determines the radius and the center of the circle to form. To determine the center, we use the resulting convex hull from Section 4.3.3. To

determine the radius, we consider two methods. In the first method, we calculate the area of the convex hull and divide the area of the convex hull by the area of one robot to get the maximum number of robots in the area. In the second method, we estimate the number of robots using the results from Section 4.3.4. Then, we take the minimum value of the results from the two methods as the estimated number of robots in the system. In this way, each root can have very high possibility to find a place on the circle boundary. The performance of the algorithm will be evaluated afterward.

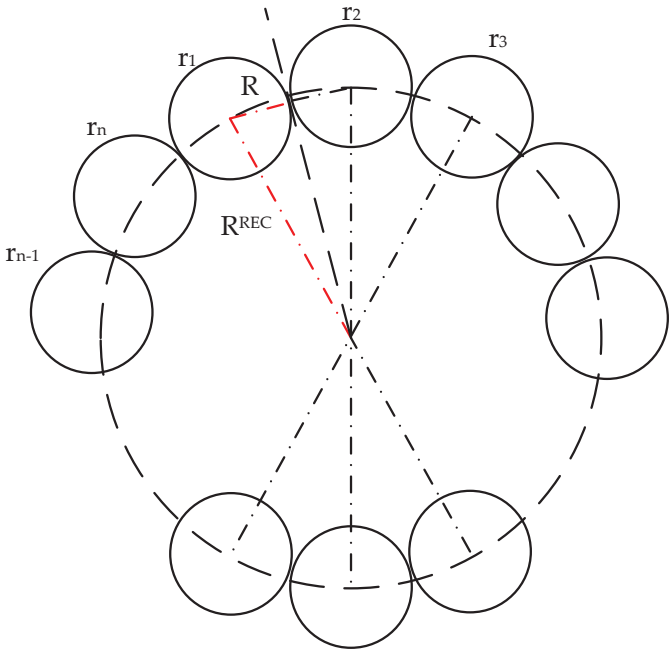


Figure 4.5 Caclulating the radius of the common circle using the estimation of the number of robots.

After estimating the number of robots, each robot can calculate the radius of the common circle as shown in Fig. 4.5. The exact algorithm is shown in Algorithm 4.

In [83], the authors give a local cardinality estimator using the theorem proved

Algorithm 4 Determination of the center and radius of the circle to be formed for each robot r_i

Input: x_a^i: parameters for cardinality estimation generated in Algorithm 3; k: parameter used in Algorithm 3; $conv_i$: convex hull generated in Algorithm 2

Output: O_i: the center of the circle; R_i^{CIR}: the radius of the circle

Begin:

1: **for** $a \leftarrow 1$ to k **do**

2: $y_a^i \leftarrow$ the number of leading continuous '1's in x_a^i

3: **end for**

4: $y \leftarrow \Sigma_{a=1}^{k} y_a^i$

5: $\hat{p}_i \leftarrow 1.2897 \times 2^{\frac{y}{k}}$ ▷ The result of distributed cardinality estimation

6: $O_i \leftarrow \frac{1}{|conv_i|}\Sigma_{j=1}^{|conv_i|} conv_i.p_j$ ▷ The center of the target circle

7: $area_i \leftarrow$ area of $conv_i$ ▷ Area of the convex hull

8: $\hat{q}_i \leftarrow \gamma \frac{area_i}{\pi R^2}$ ▷ The convex hull contains at most $\frac{area_i}{\pi R^2}$ robots. We multiply it by a parameter $\gamma \in [1, 2]$ without loss of generality.

9: $\hat{n}_i \leftarrow min(\hat{p}_i, \hat{q}_i)$

10: $R_i^{CIR} \leftarrow R / \sin \frac{\pi}{\hat{n}_i}$

11: **return** $\{O_i, R_i^{CIR}\}$

End

in [35] as follows:

$$y = \Sigma_{a=1}^{k} y_a^i$$

$$\hat{p}_i = 1.2897 \times 2^{\frac{y}{k}}$$

The performance of the estimator can be guranteed by satisfying *accuracy requirement*. The accuracy requirement is said to be satisfied if $Pr[|\hat{n} - n| \leq \beta n] \geq 1 - \alpha$, in which α is the *error probability* and β is the *confidence interval* (also called error bound). Adapting the theorem in [35] to our context, we get the constraints for k to

guarantee the performance:

Theorem 4.3.1 *Given the error probability α and confidence interval β, the accuracy requirement is satisfied if $k \geq \max\{[\frac{-\sigma_\infty c}{\log_2(1-\beta)}]^2, [\frac{\sigma_\infty c}{\log_2(1+\beta)}]^2\}$, where c is obtained by solving $1-\alpha = erf(\frac{c}{\sqrt{2}})$, erf is the Gaussian error function.*

4.3.6 Circle Formation

After the previous steps, all the robots have made a consensus on a common circle to form. Let O be the center of the circle, R^{CIR} be the radius of the circle and CIR be the common circle. In this section, Algorithm 5 is proposed to solve the circle formation problem.

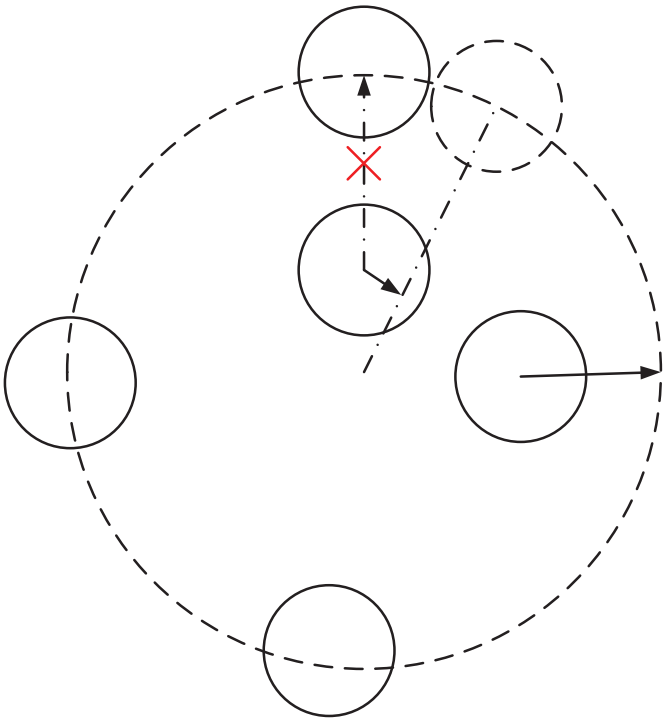


Figure 4.6 Move to the boundary of the circle

Algorithm 5 Circle Formation for each robot r_i

Input: O: the center of the circle; R^{CIR}: the radius of the circle; r_i: robot identifier; α: speed control parameter

Output: All robots are located on the boundrary of a circle

Begin:

$Cir \leftarrow$ the boundary of a circle with O as the center and R^{REC} as radius
$Ray_i \leftarrow$ the radial from O to p_i
$D_i \leftarrow$ the intersection point of O_i and Ray_i
while r_i does not reach Cir **do**
    if There is no other robot along Ray_i **then**
        Move along Ray_i to D_i
    else
        $D_{i+1} \leftarrow$ the next robot position clockwise adjacent to D_i on the Cir
        $Ray_{i+1} \leftarrow$ the ray from O to D_{i+1}
        if There is no other robots along the shortest way from p_i to Ray_{i+1} **then**
            Move along the shortest way from p_i to D_{i+1}
        else
            Wait in this round
        end if
    end if
end while

End

At the beginning, each robot r_i calculates the point D_i as the intersection point of the circle CIR and the ray Ray_i from O to p_i. And D_i serves as the target position of robot r_i. While robot r_i moves to D_i, it detects whether there is any other robot ahead along Ray_i. If so, it changes its target to the next position D_{i+1} that is clockwise

available for a robot on the circle. Fig. 4.6 shows an example of Algorithm 5, in which solid arrows are the planned routes of the robots. Since there are enough spaces to place all the robots on the circle, there must be an available position for each robot.

4.3.7 Uniform Transformation

After circle formation, the robots are going to do uniform transformation as stated in Section 4.2.2. First, the robots can be aware of the total number of robots by clockwise passing information in a straightforward way. Then, each robot calculates the desired final robot inter-distance, namely d, on the uniform circle. For robot r, r^+ is notated as the clockwise neighboring robot on the circle. We adapt the algorithm in [36] to Algorithm 6 to solve the uniform transformation problem.

Algorithm 6 Uniform transformation for each robot r_i

Input:d: the desired final robot inter-distance; $p+$: the position of r_i's successor; CIR: the circle to form

Output: All robots are uniformly located along a circle

Begin:

1: $d^+ \leftarrow distance(p_i, p^+)$

2: **if** $d^+ > d$ **then**

3: Move toward the point p on the circle CIR at distance d from p^+, remaining on the circle CIR during the movement.

4: **end if**

End

4.4 Experimental Results

To demonstrate the usefulness and evaluate the performance of our solution proposed in Section 4.3, we deploy a realistic test-bed. Our realistic test-bed, as shown in Fig. 4.7, is composed of three components, which are a localization system, multiple (currently 8) intelligent robots and a programming environment.

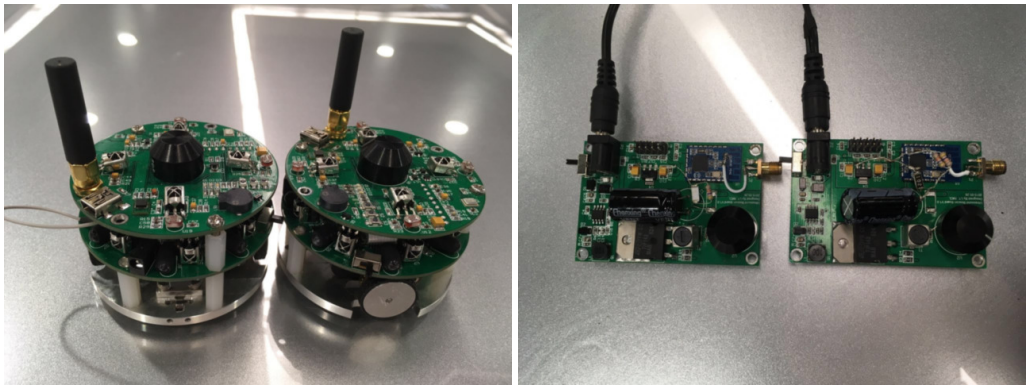


Figure 4.7 The robots and the beacons

The localization system consists of two anchor beacons, which are used for robotic localization. Each anchor beacon is composed of a 2.4G wireless communication module, an STM32 Microcontroller(MCU), an ultrasonic transmitter, a temperature sensor, and a battery. In the procedure of localization, the 2.4G wireless communication module and the ultrasonic transmitter will send signals simultaneously. Due to the different propagation velocities of the 2.4G wireless signal and the ultrasonic signal, each robot receives the two signals at different moments. As a result, each robot can calculate the distance between itself and the beacon using the principle of TDOA (time difference of arrival). After calculating the distance between the robot itself and the two anchor beacons, the robot can calculate its location on the planar platform. Since the velocity of the ultrasonic signal varies under different temperatures, we add a temperature sensor to achieve a more precise

localization.

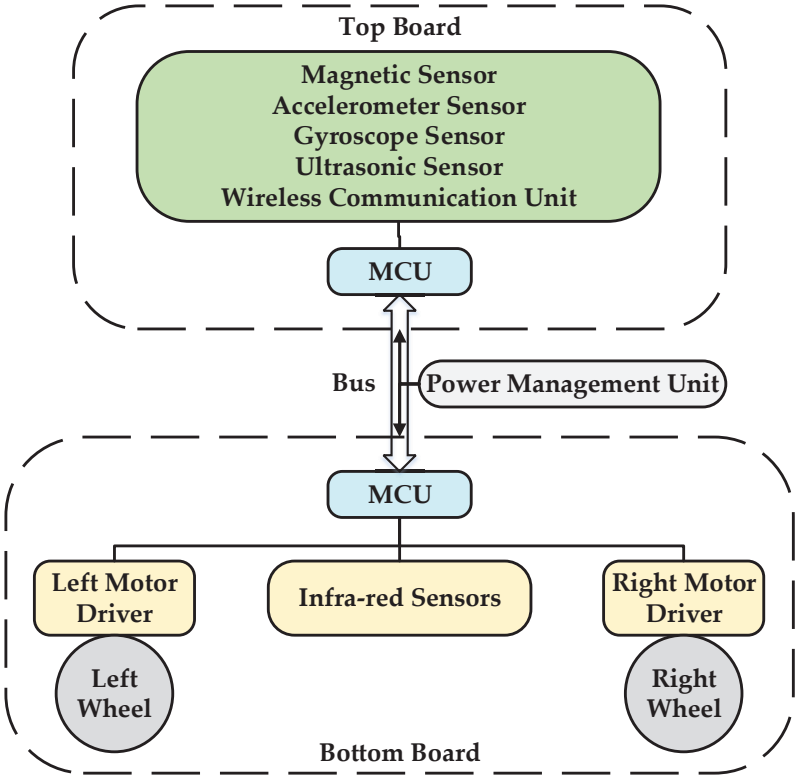


Figure 4.8 Structure Diagram of a Robot

The robots, namely PiBots (The Hong Kong Polytechnic University Intelligent Robot), are the 2nd version of our design (version 1 is presented in [51]) in our laboratory. All the robots are in the same shape and with the same size, which is approximately a cylinder with 7cm radius and 18cm height. The struct diagram of the robot is shown in Fig. 4.8. Each robot is composed of two boards, the upper board and bottom board, which are powered by the power management unit. In both of the boards, there is an MCU which is responsible for data storage and processing. Inside the MCUs, data can be stored in either 8Mbit static random access memory (SRAM) or 512Kbit flash memory. The two boards are connected by a SPI (Serial

Peripheral Interface) bus to achieve data transmission.

The bottom board includes two motor drivers, eight infra-red sensors, a motor feedback signal processor and an STM32 MCU. The motor drivers can control the wheels with given speeds separately and the motor feedback signal processor can get the current speeds of the wheels. The eight infra-red sensors can detect whether there are obstacles or not in eight directions. The upper board includes an STM32 MCU, three controlled buttons, a 2.4G wireless communication unit, a 2K EEPROM, an ultrasonic receiver, a 3-axis accelerator sensor, a 3-axis gyroscope, a 3-axis geomagnetic sensor, etc. The wireless communication unit, which uses IEEE 802.15.4 as its protocol, enables communication between robots. Other functionalities achieved by the bottom board includes localization, direction awareness, etc.

The programming environment is based on FreeRTOS, a popular real-time operating system kernel for embedded devices. In FreeRTOS, programmers can define a set of tasks (similar to threads in operating systems) with priorities to be executed concurrently. For example, in our implementation of uniform circle formation, a task to control motors, a task for communication, and a task for the purpose of user-defined application are used. Then, in every time unit, the tasks will be executed one by one according to the pre-defined priorities. Also, to simply the programming task, we utilize the programming model proposed in [49]. Finally, we deploy our algorithm for uniform circle formation in our test-bed with parameters $l = 10$, $k = 4$ and $\gamma = 1.2$. The running examples are shown in Fig. 4.1, in which scenarios with different numbers of robots and different initial configurations are considered. The results demonstrate the effectiveness and practicability of our algorithm.

4.5 Related Works

The circle formation problem was first discussed and further improved by Sugihara and Suzuki [98]. Their approach is based on heuristics and works in SSYNC for the formation of an approximate circle instead of a perfect one. Later on, researchers cast light on a particular case of circle formation called uniform circle formation in which the robots must be arranged at regular intervals on the boundary of a circle. A remarkable progress is attained by Defago and Konagaya [27]. They proposed a protocol in the SSYNC model and formally prove their approach to converge toward a uniform circle. With respect to ASYNC model, Flocchini et al. [37] addressed the uniform circle formation problem by moving to smallest enclosing circle (SEC) and avoiding *pre-regular* circumstance.

However, all the above algorithms are based on the assumptions of unlimited visibility and the punctiform hypothesis. With unlimited visibility, each robot is aware of the positions of all robots. However, robots can only sense their surroundings within a certain range in reality. With limited visibility, a robot might not even know the total number of robots. Also, assuming unlimited visibility makes this procedure unscalable and computationally expensive, since each robot has to process the information of all robots. On the other hand, robots are represented as points under the punctiform hypothesis. These assumptions greatly simplify the problem of uniform circle formation by avoiding the details of the formation of a consensus of the circle size, collision avoidance, etc.

The circumstance of limited visibility is understandably challenging. To our knowledge, only a small number of algorithmic results are known under the limited visibility scenario, even in problems other than uniform circle formation. Multi-robot gathering with limited visibility is well discussed and investigated [39][57]. On uniform circle formation, known results are conducted by Dutta et al. [32] and

Datta et al. [24]. However, both of them assume for convenience that the circle to form is given in advance. In reality, consensus on the circle is not trivial at all.

The boundary is a "tightly" wrapped contour around the configuration of robots. In this paper, we incorporate the convex hull for boundary detection. In computational geometry, the algorithm of Graham's scan [42] is well-known to compute the convex hull of a set of points in the two-dimensional space. The algorithm of Graham's scan requires the computation to be performed in a central computer, which does not meet our requirement that the robots form a distributed system and are with limited visibility. To our knowledge, only few algorithms [102][65][45] are known for distributed boundary detection. However, these algorithms only make each robot aware of whether or not it is on the boundary. Each robot is not aware of all the robots on the boundary.

Cardinality estimation, i.e., counting the approximate number of tags in a given region is widely discussed in RFID systems [54][58]. However, in existing works, only the central computation station, i.e., the RFID readers, are aware of the estimation result. To our knowledge, the problem of cardinality estimation has not been discussed yet in fully-distributed systems.

4.6 Conclusion

The problem of uniform circle formation is one of the important coordination problems in multi-robot systems. The question of whether robots with actual size and limited sensing and communication range could form a uniform circle, to our knowledge, has remained open. In this paper, we formulate the uniform circle formation problem in a distributed multi-robot system and propose a three-phase approach, namely consensus on circle, circle formation and uniform transformation towards solving it. Inside our approach, we propose several new algorithms, i.e.,

distributed convex hull construction and distributed cardinality estimation, which can be used in general distributed systems. After designing our distributed algorithm, we deploy it in our realistic test-bed and have done solid experiments. The results imply the effectiveness and practicability of our algorithms.

Bibliography

[1] Dimitris Alimisis. Educational robotics: Open questions and new challenges. *Themes in Science and Technology Education*, 6(1):63–71, 2013.

[2] Tamio Arai, Enrico Pagello, and Lynne E Parker. Editorial: Advances in multi-robot systems. *IEEE Transactions on robotics and automation*, 18(5):655–661, 2002.

[3] DJ Arbuckle and Aristides AG Requicha. Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations. *Autonomous Robots*, 28(2):197–211, 2010.

[4] Michael P Ashley-Rollman, Seth Copen Goldstein, Peter Lee, Todd C Mowry, and Padmanabhan Pillai. Meld: A declarative approach to programming ensembles. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2794–2800. IEEE, 2007.

[5] Michael P Ashley-Rollman, Peter Lee, Seth Copen Goldstein, Padmanabhan Pillai, and Jason D Campbell. A language for large ensembles of independently executing nodes. In *Logic Programming*, pages 265–280. Springer, 2009.

[6] Jonathan Bachrach and Jacob Beal. Programming a sensor network as an amorphous medium. 2006.

[7] Jonathan Bachrach, James McLurkin, and Anthony Grue. Protoswarm:

a language for programming multi-robot systems using the amorphous medium abstraction. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1175–1178. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[8] Richard Barry. *FreeRTOS reference manual: API functions and configuration options*. Real Time Engineers Limited, 2009.

[9] Jacob Beal and Jonathan Bachrach. Infrastructure for engineered emergence on sensor/actuator networks. *Intelligent Systems, IEEE*, 21(2):10–19, 2006.

[10] Ryan A Beasley. Medical robots: current systems and research directions. *Journal of Robotics*, 2012, 2012.

[11] Fabiane Barreto Vavassori Benitti. Exploring the educational potential of robotics in schools: A systematic review. *Computers & Education*, 58(3):978–988, 2012.

[12] Elizabeth Broadbent, Rebecca Stafford, and Bruce MacDonald. Acceptance of healthcare robots for the older population: review and future directions. *International Journal of Social Robotics*, 1(4):319–330, 2009.

[13] James Bruce, Stefan Zickler, Mike Licitra, and Manuela Veloso. Cmdragons: Dynamic passing and strategy on a champion robot soccer team. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 4074–4079. IEEE, 2008.

[14] Wolfram Burgard, Mark Moors, Dieter Fox, Reid Simmons, and Sebastian Thrun. Collaborative multi-robot exploration. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 476–481. IEEE, 2000.

[15] Jessica Burgner-Kahrs, D Caleb Rucker, and Howie Choset. Continuum

robots for medical applications: A survey. *IEEE Transactions on Robotics*, 31(6):1261–1280, 2015.

[16] William Joseph Butera. *Programming a paintable computer*. PhD thesis, Citeseer, 2002.

[17] Stephan K Chalup, Craig L Murch, and Michael J Quinlan. Machine learning with aibo robots in the four-legged league of robocup. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(3):297–310, 2007.

[18] Ioannis Chatzigiannakis, Michael Markou, and Sotiris Nikoletseas. Distributed circle formation for anonymous oblivious robots. In *International Workshop on Experimental and Efficient Algorithms*, pages 159–174. Springer, 2004.

[19] David Chu, Arsalan Tavakoli, Lucian Popa, and Joseph Hellerstein. Entirely declarative sensor network systems. In *Proceedings of the 32nd international conference on Very large data bases*, pages 1203–1206. VLDB Endowment, 2006.

[20] Christopher M Cianci, Xavier Raemy, Jim Pugh, and Alcherio Martinoli. Communication in a swarm of miniature robots: The e-puck as an educational tool for swarm robotics. In *International Workshop on Swarm Robotics*, pages 103–115. Springer, 2006.

[21] William Cook and Andre Rohe. Computing minimum-weight perfect matchings. *INFORMS Journal on Computing*, 11(2):138–148, 1999.

[22] Flavio Cruz12, Ricardo Rocha, and Seth Copen Goldstein. A parallel virtual machine for executing forward-chaining linear logic programs. In *Workshop on Implementation of Constraint and Logic Programming Systems and Logic-based Methods in Programming Environments 2014*, page 125.

[23] Leonardo Dagum and Rameshm Enon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[24] Suparno Datta, Ayan Dutta, Sruti Gan Chaudhuri, and Krishnendu Mukhopadhyaya. Circle formation by asynchronous transparent fat robots. In *International Conference on Distributed Computing and Internet Technology*, pages 195–207. Springer, 2013.

[25] Michael De Rosa, Seth Goldstein, Peter Lee, Jason Campbell, and Padmanabhan Pillai. Scalable shape sculpting via hole motion: Motion planning in lattice-constrained modular robots. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1462–1468. IEEE, 2006.

[26] Michael De Rosa, Seth Goldstein, Peter Lee, Padmanabhan Pillai, and Jason Campbell. Programming modular robots with locally distributed predicates. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3156–3162. IEEE, 2008.

[27] Xavier Défago and Akihiko Konagaya. Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 97–104. ACM, 2002.

[28] Dominique Dhoutaut, Benoît Piranda, and Julien Bourgeois. Efficient simulation of distributed sensing and control environments. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 452–459. IEEE, 2013.

[29] Ezequiel Di Mario and Alcherio Martinoli. Distributed particle swarm opti-

mization for limited-time adaptation with real robots. *Robotica*, 32(02):193–208, 2014.

[30] Yoann Dieudonné, Ouiddad Labbani-Igbida, and Franck Petit. Circle formation of weak mobile robots. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 3(4):16, 2008.

[31] Yoann Dieudonné and Franck Petit. Squaring the circle with weak mobile robots. In *International Symposium on Algorithms and Computation*, pages 354–365. Springer, 2008.

[32] Ayan Dutta, Sruti Gan Chaudhuri, Suparno Datta, and Krishnendu Mukhopadhyaya. Circle formation by asynchronous fat robots with limited visibility. In *International Conference on Distributed Computing and Internet Technology*, pages 83–93. Springer, 2012.

[33] Joseph F Engelberger. *Robotics in practice: management and applications of industrial robots*. Springer Science & Business Media, 2012.

[34] Alessandro Farinelli, Luca Iocchi, and Daniele Nardi. Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2015–2028, 2004.

[35] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[36] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Self-deployment algorithms for mobile sensors on a ring. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 59–70. Springer, 2006.

[37] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Distributed computing by mobile robots: uniform circle formation. *Dis-*

tributed Computing, pages 1–45, 2014.

[38] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In *Algorithms and Computation, 10th International Symposium, ISAAC'99, Chennai, India, December 16-18, 1999, Proceedings*, pages 93–102, 1999.

[39] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1-3):147–168, 2005.

[40] Nao Fujinaga, Yukiko Yamauchi, Hirotaka Ono, Shuji Kijima, and Masafumi Yamashita. Pattern formation by oblivious asynchronous mobile robots. *SIAM Journal on Computing*, 44(3):740–785, 2015.

[41] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jerome Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. The nao humanoid: a combination of performance and affordability. *CoRR abs/0807.3223*, 2008.

[42] Ronald L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information processing letters*, 1(4):132–133, 1972.

[43] Luigi Alfredo Grieco, Alessandro Rizzo, S Colucci, Sabrina Sicari, Giuseppe Piro, Donato Di Paola, and Gennaro Boggia. Iot-aided robotics applications: Technological implications, target domains and open issues. *Computer Communications*, 54:32–47, 2014.

[44] Ramakrishna Gummadi, Omprakash Gnawali, and Ramesh Govindan. Macro-programming wireless sensor networks using kairos. In *International Conference on Distributed Computing in Sensor Systems*, pages 126–140. Springer, 2005.

[45] Peng Guo, Jiannong Cao, and Kui Zhang. Distributed topological convex hull estimation of event region in wireless sensor networks without location information. *IEEE transactions on parallel and distributed systems*, 26(1):85–94, 2015.

[46] Golnaz Habibi, William Xie, Mathew Jellins, and James McLurkin. Distributed path planning for collective transport using homogeneous multi-robot systems. In *Distributed Autonomous Robotic Systems*, pages 151–164. Springer, 2016.

[47] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *ACM SIGOPS operating systems review*, volume 34, pages 93–104. ACM, 2000.

[48] Andrew Howard, Lynne E Parker, and Gaurav S Sukhatme. Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *The International Journal of Robotics Research*, 25(5-6):431–447, 2006.

[49] Shan Jiang, Jiannong Cao, Yang Liu, Jinlin Chen, and Xuefeng Liu. Programming large-scale multi-robot system with timing constraints. In *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*, pages 1–9. IEEE, 2016.

[50] Shan Jiang, Jiannong Cao, Jia Wang, Milos Stojmenovic, and Julien Bourgeois. Uniform circle formation by asynchronous robots: A fully-distributed approach. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9. IEEE, 2017.

[51] Shan Jiang, Junbin Liang, Jiannong Cao, and Rui Liu. An ensemble-level programming model with real-time support for multi-robot systems. In *2016*

IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops), pages 1–3. IEEE, 2016.

[52] Serge Kernbach, Ronald Thenius, Olga Kernbach, and Thomas Schmickl. Re-embodiment of honeybee aggregation behavior in an artificial micro-robotic system. *Adaptive Behavior*, 17(3):237–259, 2009.

[53] Sangbae Kim, Cecilia Laschi, and Barry Trimmer. Soft robotics: a bioinspired evolution in robotics. *Trends in biotechnology*, 31(5):287–294, 2013.

[54] Murali Kodialam and Thyaga Nandagopal. Fast and reliable estimation schemes in rfid systems. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 322–333. ACM, 2006.

[55] Nupur Kothari, Ramakrishna Gummadi, Todd Millstein, and Ramesh Govindan. Reliable and efficient programming abstractions for wireless sensor networks. In *ACM SIGPLAN Notices*, volume 42, pages 200–210. ACM, 2007.

[56] Pedro U Lima and Luis M Custodio. Multi-robot systems. In *Innovations in robot mobility and control*, pages 1–64. Springer, 2005.

[57] Jie Lin, A Stephen Morse, and Brian DO Anderson. The multi-agent rendezvous problem. part 2: The asynchronous case. *SIAM Journal on Control and Optimization*, 46(6):2120–2147, 2007.

[58] Xiulong Liu, Keqiu Li, Jie Wu, Alex X Liu, Xin Xie, Chunsheng Zhu, and Weilian Xue. Top-k queries for multi-category rfid systems. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[59] Boon Thau Loo, Tyson Condie, Minos Garofalakis, David E Gay, Joseph M Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking: language, execution and optimization. In *Proceedings of the 2006 ACM SIGMOD international conference on Man-*

agement of data, pages 97–108. ACM, 2006.

[60] Yuri K Lopes, André B Leal, Tony J Dodd, and Roderich Groß. Application of supervisory control theory to swarms of e-puck and kilobot robots. In *International Conference on Swarm Intelligence*, pages 62–73. Springer, 2014.

[61] Lingzhi Luo, Nilanjan Chakraborty, and Katia Sycara. Provably-good distributed algorithm for constrained multi-robot task assignment for grouped tasks. *Robotics, IEEE Transactions on*, 31(1):19–30, 2015.

[62] Samuel R Madden, Michael J Franklin, Joseph M Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on database systems (TODS)*, 30(1):122–173, 2005.

[63] Norman Margolus. Cam-8: a computer architecture based on cellular automata. *Pattern Formation and Lattice-Gas Automata*, pages 167–187, 1996.

[64] Maja J Mataric. Interaction and intelligent behavior. Technical report, DTIC Document, 1994.

[65] James McLurkin and Erik D Demaine. A distributed boundary detection algorithm for multi-robot systems. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4791–4798. IEEE, 2009.

[66] James McLurkin, Andrew J Lynch, Scott Rixner, Thomas W Barr, Alvin Chou, Kathleen Foster, and Siegfried Bilstein. A low-cost multi-robot system for research, teaching, and outreach. In *Distributed Autonomous Robotic Systems*, pages 597–609. Springer, 2013.

[67] James McLurkin, Adam McMullen, Nick Robbins, Golnaz Habibi, Aaron Becker, Alvin Chou, Hao Li, Meagan John, Nnena Okeke, Joshua Rykowski, et al. A robot system design for low-cost multi-robot manipulation. In *2014*

IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 912–918. IEEE, 2014.

[68] James McLurkin, Joshua Rykowski, Meagan John, Quillan Kaseman, and Andrew J Lynch. Using multi-robot systems for engineering education: Teaching and outreach with large numbers of an advanced, low-cost robot. *IEEE transactions on education*, 56(1):24–33, 2013.

[69] James McLurkin and Jennifer Smith. Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots. In *in 7th International Symposium on Distributed Autonomous Robotic Systems (DARS*. Citeseer, 2004.

[70] James McLurkin, Jennifer Smith, James Frankel, David Sotkowitz, David Blau, and Brian Schmidt. Speaking swarmish: Human-robot interface design for large swarms of autonomous mobile robots. In *AAAI Spring Symposium: To Boldly Go Where No Human-Robot Team Has Gone Before*, pages 72–75, 2006.

[71] Nathan Michael, Jonathan Fink, and Vijay Kumar. Experimental testbed for large multirobot teams. *IEEE robotics & automation magazine*, 15(1):53–61, 2008.

[72] Nelson Minar, Roger Burkhart, Chris Langton, Manor Askenazi, et al. The swarm simulation system: A toolkit for building multi-agent simulations. Santa Fe Institute Santa Fe, 1996.

[73] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stephane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and Alcherio Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, volume 1, pages 59–65. IPCB: Instituto

Politécnico de Castelo Branco, 2009.

[74] Francesco Mondada, Edoardo Franzi, and Andre Guignard. The development of khepera. In *Experiments with the Mini-Robot Khepera, Proceedings of the First International Khepera Workshop*, number LSRO-CONF-2006-060, pages 7–14, 1999.

[75] Ryan Newton, Greg Morrisett, and Matt Welsh. The regiment macroprogramming system. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 489–498. ACM, 2007.

[76] Ulf Nilsson and Jan Małuszyński. *Logic, programming and Prolog*. Wiley Chichester, 1990.

[77] Kwang-Kyo Oh, Myoung-Chul Park, and Hyo-Sung Ahn. A survey of multi-agent formation control. *Automatica*, 53:424–440, 2015.

[78] Gina Owens, Yael Granader, Ayla Humphrey, and Simon Baron-Cohen. Lego® therapy and the social use of language programme: An evaluation of two social skills interventions for children with high functioning autism and asperger syndrome. *Journal of autism and developmental disorders*, 38(10):1944–1957, 2008.

[79] Dimitra Panagou, Dušan M Stipanović, and Petros G Voulgaris. Distributed coordination control for multi-robot networks using lyapunov-like barrier functions. *IEEE Transactions on Automatic Control*, 61(3):617–632, 2016.

[80] Lynne E Parker. Current state of the art in distributed autonomous mobile robotics. In *Distributed Autonomous Robotic Systems 4*, pages 3–12. Springer, 2000.

[81] Giuseppe Prencipe and Nicola Santoro. Distributed algorithms for autonomous mobile robots. In *Fourth IFIP International Conference on Theoretical Computer Science-TCS 2006*, pages 47–62. Springer, 2006.

[82] Jim Pugh, Xavier Raemy, Cedric Favre, Riccardo Falconi, and Alcherio Martinoli. A fast onboard relative positioning module for multirobot systems. *IEEE/ASME Transactions on Mechatronics*, 14(2):151–162, 2009.

[83] Chen Qian, Hoilun Ngan, Yunhao Liu, and Lionel M Ni. Cardinality estimation for large-scale rfid systems. *IEEE transactions on parallel and distributed systems*, 22(9):1441–1454, 2011.

[84] John G Rogers III, Alexander JB Trevor, Carlos Nieto-Granda, Alex Cunningham, Manohar Paluri, Nathan Michael, Frank Dellaert, Henrik I Christensen, and Vijay Kumar. Effects of sensory precision on mobile robot localization and mapping. In *Experimental Robotics*, pages 433–446. Springer, 2014.

[85] Michael Rubenstein, Christian Ahler, Nick Hoff, Adrian Cabrera, and Radhika Nagpal. Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, 62(7):966–975, 2014.

[86] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3293–3298. IEEE, 2012.

[87] Michael Rubenstein, Adrian Cabrera, Justin Werfel, Golnaz Habibi, James McLurkin, and Radhika Nagpal. Collective transport of complex objects by simple robots: theory and experiments. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 47–54. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[88] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Pro-

grammable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[89] Michael Rubenstein and Wei-Min Shen. Automatic scalable size selection for the shape of a distributed robotic collective. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 508–513. IEEE, 2010.

[90] Sajad Saeedi, Michael Trentini, Mae Seto, and Howard Li. Multiple-robot simultaneous localization and mapping: A review. *Journal of Field Robotics*, 33(1):3–46, 2016.

[91] Yuvraj Sahni, Jiannong Cao, and Shan Jiang. Middleware for multi-robot systems. In *Mission-oriented sensor networks and systems: Art and science*, pages 633–673. Springer, 2019.

[92] P Sapaty. Military robotics: Latest trends and spatial grasp solutions. *International Journal of Advanced Research in Artificial Intelligence*, 4(4):9–18, 2015.

[93] Guillaume Sartoretti, Max-Olivier Hongler, Marcelo Elias de Oliveira, and Francesco Mondada. Decentralized self-selection of swarm trajectories: from dynamical systems theory to robotic implementation. *Swarm Intelligence*, 8(4):329–351, 2014.

[94] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer Science & Business Media, 2008.

[95] Jorge M Soares, A Pedro Aguiar, António M Pascoal, and Alcherio Martinoli. A graph-based formation algorithm for odor plume tracing. In *Distributed Autonomous Robotic Systems*, pages 255–269. Springer, 2016.

[96] Jorge M Soares, Inaki Navarro, and Alcherio Martinoli. The khepera iv mobile robot: Performance evaluation, sensory data and software toolbox. In

Robot 2015: Second Iberian Robotics Conference, pages 767–781. Springer, 2016.

[97] Kasper Stoy and Radhika Nagpal. Self-repair through scale independent self-reconfiguration. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 2, pages 2062–2067. IEEE, 2004.

[98] Kazuo Sugihara and Ichiro Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *Journal of robotic systems*, 13(3):127–139, 1996.

[99] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots. In *SIROCCO'96, The 3rd International Colloquium on Structural Information & Communication Complexity, Siena, Italy, June 6-8, 1996*, pages 313–330, 1996.

[100] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

[101] Katherine M Tsui and Holly A Yanco. Assistive, rehabilitation, and surgical robots from the perspective of medical and healthcare professionals. In *AAAI 2007 Workshop on Human Implications of Human-Robot Interaction, Technical Report WS-07-07 Papers from the AAAI 2007 Workshop on Human Implications of HRI*, 2007.

[102] Yue Wang, Jie Gao, and Joseph SB Mitchell. Boundary recognition in sensor networks by topological methods. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 122–133. ACM, 2006.

[103] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler. Hood: a

neighborhood abstraction for sensor networks. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 99–110. ACM, 2004.

[104] L Elena Whittier and Michael Robinson. Teaching evolution to non-english proficient students by using lego robotics. *American Secondary Education*, pages 19–28, 2007.

[105] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9, 2008.

[106] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10, 2013.