This document is downloaded from DR-NTU, Nanyang Technological University Library, Singapore.

Title	Distributed approaches to motion planning and control in multi-robot systems
Author(s)	Zhou, Yuan
Citation	Zhou, Y. (2019). Distributed approaches to motion planning and control in multi-robot systems. Doctoral thesis, Nanyang Technological University, Singapore.
Date	2019-05-14
URL	http://hdl.handle.net/10220/48184
Rights	

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

DISTRIBUTED APPROACHES TO MOTION PLANNING AND CONTROL IN MULTI-ROBOT SYSTEMS

YUAN ZHOU

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

2019

DISTRIBUTED APPROACHES TO MOTION PLANNING AND CONTROL IN MULTI-ROBOT SYSTEMS

YUAN ZHOU

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

12 April 2019	周远
. . . .	. . . .
Date	Yuan Zhou

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

15 April 2019

. . . .

Date

. . . .

Shang-Wei Lin

Authorship Attribution Statement

This thesis contains materials from 3 papers published in the following peer-reviewed journals for which I am the first author.

Chapter 4 is published as Yuan Zhou, Hesuan Hu, Yang Liu, Shang-Wei Lin, Zuohua Ding, "A real-time and fully distributed approach to motion planning for multirobot systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Oct. 2017. http://ieeexplore.ieee.org/document/8055437/. DOI: 10.1109/TSMC.2017.2750911

The contributions of the co-authors are as follows:

- I was the lead author. I wrote the manuscript drafts and conducted all experiments.
- Prof Hu guided the initial research direction and revised the manuscript drafts.
- I co-designed the methodology with Prof Hu.
- Profs Liu, Lin, and Ding discussed and supported the research, and revised the drafts.

Chapters 6 is published as Yuan Zhou, Hesuan Hu, Yang Liu, and Zuohua Ding, "Collision and deadlock avoidance in multirobot systems: A distributed approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 7, pp. 1712–1726, Jul. 2017. DOI: 10.1109/TSMC.2017.2670643

The contributions of the co-authors are as follows:

- I was the lead author. I wrote the manuscript drafts and conducted all experiments.
- Prof Hu guided the initial research direction and revised the manuscript drafts.

- I co-designed the methodology with Prof Hu.
- Profs Liu, Lin, and Ding discussed and supported the research, and revised the drafts.

Chapter 8 is published as Yuan Zhou, Hesuan Hu, Yang Liu, Shang-Wei Lin, and Zuohua Ding, “A distributed approach to robust control of multi-robot systems,” *Automatica*, vol. 98, pp. 1–13, Dec. 2018. DOI: 10.1016/j.automatica.2018.08.022

The contributions of the co-authors are as follows:

- I was the lead author. I wrote the manuscript drafts and conducted all experiments.
- Prof Hu guided the initial research direction and revised the manuscript drafts.
- I co-designed the methodology with Prof Hu.
- Profs Liu, Lin, and Ding discussed and supported the research, and revised the drafts.

12 April 2019	周远
. . . .	. . . .
Date	Yuan Zhou

Acknowledgements

As time flies, it is near the end of my PhD study. Though the PhD life fulfills pain and happiness, I gain a lot from the study: knowledge, skills, love, friendship, etc. No words can fully express myself, and all I can say is thank you, thank you very much.

First, I would like to extend my sincere gratitude to my supervisors Profs. Lin Shang-Wei, Liu Yang, and Hu Hesuan for their valuable guidance and advice. Their profound knowledge, rich experience, and sagacious perception encourage me to focus on the right research field. Their detailed guidance helps me to improve my skills for thinking and writing greatly. Their invaluable support throughout these years has really helped me to conduct my research. Without them, I cannot finish my study or this thesis.

Second, I would like to extend my grateful thanks to Prof. Ding Zuohua, who is the enlightenment teacher of my academic career. He always gives me his suggestions and encouragement on both research and life.

Third, I would like to thank my friends and colleague in cyber security lab (CSL) for their help in both my life and study, especially Dr Chen Bihuan, Dr Meng Guozhu, Dr Wang Haijun, Dr Xie Xiaofei, Dr Xue Yinxing, Cheng Kun, Chen Hongxu, Du Xiaoning, Feng Ruitao, Li Yuekang, Tang Yun, Wang Junjie, Xu Zhengzi, and so on. I also thank our laboratory executive Tan SuanHai for his kind help to buy and setup the hardware of UAVs, as well as provide IT support.

Last but the most important, I would like to express my utmost appreciation to my family: my parents, my wife, and my elder sisters. I'm heavily indebted to them for their unconditional support to my study. Without their support and sacrifice, I also cannot finish my study.

Contents

List of Figures

List of Tables

Summary

A multi-robot system is a system containing multiple robots which are moving around a given environment to accomplish tasks cooperatively. Motion planning and control is one of the most important issues in multi-robot systems. On one hand, as an individual, a robot is expected to make collision-free motion under its own controller; on the other hand, as a whole system, a robot is expected to move cooperatively with others. Hence, in this thesis, we focus on distributed approaches to motion planning and control, which not only guarantee flexibility and scalability of the systems, but also allow negotiation among robots.

First of all, based on the kinematic equations of robots, we focus on distributed trajectory planning for multi-robot systems operating in an unstructured environment. We propose a fully distributed approach to planning trajectories, which means that a robot can compute its trajectory and perform its motion in a distributed way. It combines model predictive control (MPC) strategy and incremental sequential convex programming (iSCP) method. On each prediction horizon, a robot builds a non-convex programming by communicating with its neighbors to retrieve their current states. Based on the retrieved information, the robot predicts its neighbors' future positions by itself, so it does not need to wait for information predicted by other robots. Each robot solves its own local problem independently via the iSCP method. Once the computation is finished, the robot can move independently. Hence, the proposed method is fully distributed.

Second, with the paths generated from path/trajectory planning, we study a distributed approach to collision and deadlock avoidance in multi-robot systems where each robot has a predetermined path. We propose a real-time and distributed algorithm for collision and deadlock avoidance by repeatedly stopping and resuming robots. The motion of each robot is first modeled as a labeled transition system (LTS) and then controlled by a distributed algorithm to avoid collisions and deadlocks. Each robot can execute its local algorithm by checking whether its succeeding state is occupied and whether the one-step move can cause deadlocks. Performance analysis of the proposed algorithm is also conducted. The conclusion is that the algorithm is not only practically operative but also maximally permissive in terms of the LTS models.

Third, aiming at some more complex path networks, we further study a distributed approach to avoiding higher-order deadlocks, from which a system leads to a deadlock inevitably. Based on the LTS models of robots, we conclude that there exist at most the $(N-3)$-th order deadlocks with N robots. This means that deadlocks, if any, will occur unavoidably within $N-3$ steps of corresponding transitions. A distributed algorithm is then proposed to avoid higher-order deadlocks. To execute its local algorithm, a robot, on one hand, needs to look ahead at most $N-1$ states, i.e., $N-3$ intermediate states and two endpoint states, to check the status of these states; on the other hand, it needs to communicate with others via a multi-hop communication path to determine whether there are any circuits. By analyzing the returned circuits independently, the robot can determine whether there exist higher-order deadlocks. Theoretical analysis and experimental study show that the proposed algorithm is practically operative.

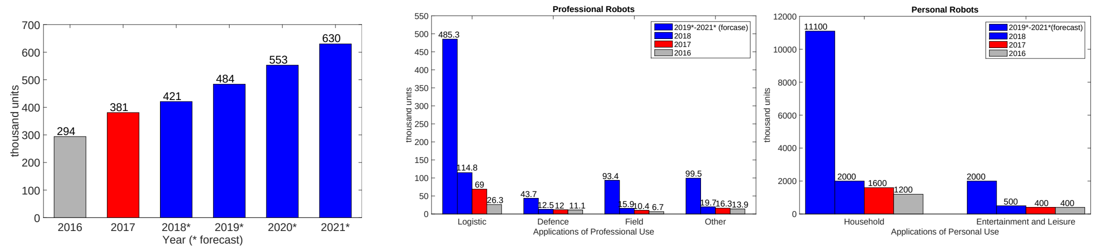
Fourth, considering the factor that there may exist robot failures in a system, we in the sequel study robust control for a multi-robot system. We classify robots into reliable and unreliable ones and assume reliable robots can always work well and unreliable ones may fail unpredictably. The objective of our robust control is to minimize the adverse effect of a failed robot on the whole system. During the path/trajectory planning phase, robust control can be done easily by regarding the failed robots as static obstacles. So we focus on systems with fixed paths. Based on the LTS models, we propose two distributed robust control algorithms: one for reliable robots and the other for unreliable ones. The algorithms guarantee that wherever an unreliable robot fails, only the robots are blocked whose state spaces contain the failed state. Theoretical analysis shows that the proposed algorithms are practically operative. Simulation results show the effectiveness of our algorithms.

Finally, to generate continuous inputs directly during deadlock avoidance, we concentrate on a distributed and hybrid approach, combining both continuous and discrete technologies studied before, to motion control for multi-robot systems where each robot has a fixed path. Based on MPC strategy, on each horizon, the discrete control part determines a proper waiting decision based on the discrete models to avoid collisions and deadlocks; then the continuous part computes proper continuous inputs by constructing and resolving a local optimization problem, which includes the constraint of waiting time. The advantages of the proposed hybrid approach are that: discrete control can deal with deadlocks and reduce the scale of optimization problem; continuous control can general optimal speed satisfying the discrete decision. In the proposed approach, to move in a fully distributed way, each robot needs to communicate with its neighbors to retrieve their current states, which can be obtained immediately. The communication protocols are described in Petri nets, and the communication network can be reconfigured in real time based on the connectivity among robots.

Chapter 1

Introduction

Since the development of the first mobile and intelligent robot **Shakey** between 1966 and 1972, mobile robots become increasingly popular and have applications in different areas, such as environment monitoring [1], search and rescue [2], reconnaissance and surveillance missions [3], demining [4], and domestic service [5]. On one hand, robots can help people do labor-consuming tasks. This can liberate us from heavy manual labor so that we can do some more creative tasks. For example, robots can help people complete assemble parts, clean houses, and mow lawns. On the other hand, mobile robots can help to do the dangerous tasks or the tasks that people cannot complete currently, such as search and rescue, humanitarian demining, underwater exploration, and space exploration. In the past years, the number of robots deployed in industries and our daily life is increasing considerably and impressively. As reported by the International Federation of Robotics (IFR) in 2018, the amount of annual worldwide supply of industrial and service robots reaches a new peak due to the rapid growth in 2017: industrial robot sales increased by 30% to 381,335 units, the number of professional service robots sold rose by 85% to 109,543 units, and that for personal and domestic use increased by 25% to about 8.5 million units. Based on their prediction, the estimated annual worldwide supply of industrial robots will be 421, 484, 553, and 630 thousand units in years 2018 − 2021, respectively; the estimated worldwide supplies of professional and personal robots in 2018 are 162.9 thousand units and 2.5 million units, respectively, while their cumulative numbers from 2019 to 2021 are 721.9 thousand units and 13.1 million units,



(a) Industrial robots. (b) Service robots.

FIG. 1.1: Statistic data of worldwide annual supply of industrial and service robots. (a) Worldwide supply of industrial robots in different years. (b) Worldwide supply of service robots with different applications.

respectively. Fig. 1.1 shows the numbers of industrial and service robots that are supplied in 2016 and 2017, and the predictions from 2018 to 2021. The data are from the report given by IFR [6].

1.1 Motivations and Challenges

Even though the last few decades witness the rapid development in robotics, the applicability of autonomous robots, like unmanned ground vehicles (UGVs) and unmanned aerial vehicles (UAVs), is still limited in our daily life due to the lack of everlasting safety guarantees during their motion in complex environments. What's more, most of the current applications are single-robot systems, i.e., each task is completed by only one robot. With the development of technologies and society, people are facing more and more complicated tasks. Thus, a single robot cannot finish these tasks efficiently. This arouses our study on motion planning and control for multi-robot systems.

A multi-robot system is a system containing multiple mobile robots that work together to complete sophisticated tasks by moving around in a given environment. The main characteristic of a multi-robot system is the cooperation among robots. Compared with their single-robot counterparts, multi-robot systems become increasingly popular thanks to their great benefits [7, 8], such as:

- Wide coverage and diverse functionality. Robots in a multi-robot system can be deployed in a wide region. Thus, the system can cover large space, do different tasks, and collect different data through the physically distributed sensors and actuators.

- High reliability and good flexibility. With multiple robots in it, a system has some redundancy. When a robot is failed, others may still cooperate to finish the tasks. With the cooperation of multiple robots, the design of each robot can be simple. Hence, the design of a multi-robot system can be more flexible.

- High performance. By decomposing a complicated task into a set of simple subtasks, which can be finished by each robot, a multi-robot system can fulfill complicated tasks and improve system performance.

All the above aspects attract our attention to multi-robot systems. Motion planning and control is one of the most critical and important issues in multi-robot systems and has been given wide consideration in both academia and industry. However, it is not a straightforward task due to the following challenges.

1. The general motion planning problems are hard to solve. Some theoretical research work has characterized the complexity of motion planning problems. Reif [9] first showed that the generalized mover's problem, i.e, finding a collision-free path for a rigid body, which may consist of multiple polyhedra, such that it can move from an initial position to a target position in a Euclidean space with polyhedral obstacles, is PSPACE-hard in 3-D space. Hopcroft *et al* [10] further showed that even for a simplified 2-D case, the coordinated motion planning problem is PSPACE-hard, which can be described as: given a set of disjoint rectangular objects and their initial and final positions in a 2-D rectangular box, plan a continuous coordinated motion such that each object can move from its initial position to the final one without causing collisions with the box and others. Besides, Reif and Sharir [11] showed that solving motion problems in dynamic environments is much harder than in static environment in terms of computational complexity. Thus, there is no efficient algorithm to solve general motion planning problems. This directs researchers to identify special cases or to find more practical approximate methods.

2. Dynamic and complex environments require real-time motion planning. Since there are multiple robots moving in the same environment, it introduces new challenges to control robot motion. On one hand, as usual, a robot needs to avoid collisions with environmental obstacles, such as people, houses, walls, chairs, and desks. Usually,

the environment is unstructured, and the obstacles are with arbitrary shapes. On the other hand, a robot needs to avoid collisions with other robots. To guarantee its motion independence so as to leverage the advantages of multi-robot systems, each robot has an individual controller and regards other robots as dynamic obstacles. In a changing environment, a robot needs to plan its motion in real time.

3. Flexibility and cooperation require distributed control. In a multi-robot system, as an individual, each robot is preferred to move somehow independently so as to keep flexibility; while as a whole, a robot is required to keep consistence with others to finish cooperative motion. Usually, the control of a multi-robot system admits centralized, decentralized, or distributed architecture. A centralized controller guides the motion of all robots simultaneously with the highest performance; however, centralized control usually lacks of flexibility and robustness. Decentralized control allows a robot to have its individual local controller, which can guarantee flexibility of the system, but it is hard for cooperation since there is no communication among robots. In distributed control, each robot has an individual local controller and different controllers can communicate with each other. Hence, to achieve cooperation among robots and guarantee flexibility of the system, distributed approaches are required to control robot motion in multi-robot systems.

4. Deadlocks may occur during distributed motion. Since robots move in a distributed manner, deadlocks may occur during the evolution of a multi-robot system. Moreover, in some cases, e.g., robots are required to move along predefined paths with multiple successive intersections, deadlocks are hard to predict since even though the system is deadlock-free currently, it will lead to a deadlock inevitably. Hence, efficient methods for deadlock prediction and avoidance are necessary and indispensable.

5. Robot failures may occur to stagnate the whole system. In a multi-robot system, some robots may fail unexpectedly during their motion. A failed robot may block the motion of the normal ones, even though some blockage can be avoided. Thus, a well-designed motion control algorithm should be robust against robot failures, i.e., minimize the number of robots that are blocked.

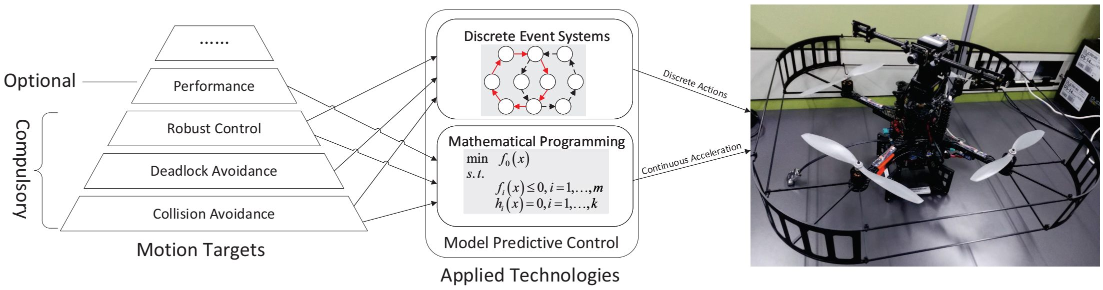


FIG. 1.2: Motion requirements and applied technologies.

1.2 Main Work

Facing the above motivations and challenges, we focus on distributed approaches to motion planning and control of multi-robot systems, and this thesis would like to answer the following questions: (1) how can each robot in a multi-robot system, which is deployed in an unstructured environment, *plan its trajectory* in a distributed way? (2) after each robot obtains its path from trajectory/path planning, how can it move along a given path in a distributed manner, *avoiding collisions and deadlocks*? (3) how can the motion of a robot be *robust against robot failures* in the system?

Generally, as shown in Fig. 1.2, the requirements for the motion of a robot contain collision avoidance, deadlock avoidance, robustness, and performance optimization. The first and fundamental level is collision avoidance. This is the most important and common requirement for safe motion of a robot. A collision not only affects the completion of motion tasks but also collapses robots. The second level is deadlock avoidance. Deadlocks may occur during collision avoidance and stop the motion of robots. A deadlock will degrade the performance of the system and make some motion tasks impossible, but all robots can still perform well once deadlocks are resolved. The third level is robust control. This is required only when there are robot failures. When a robot fails, it may block the motion of some robots, which may in turn block others. Hence, for robust control, we would like to minimize the detrimental effects of robot failures on others. All these three levels focus on the functionality of a system to finish its assigned motion tasks and should be always satisfied. These are the basic three levels, i.e., behavior implementation, and they are compulsory. When a robot can

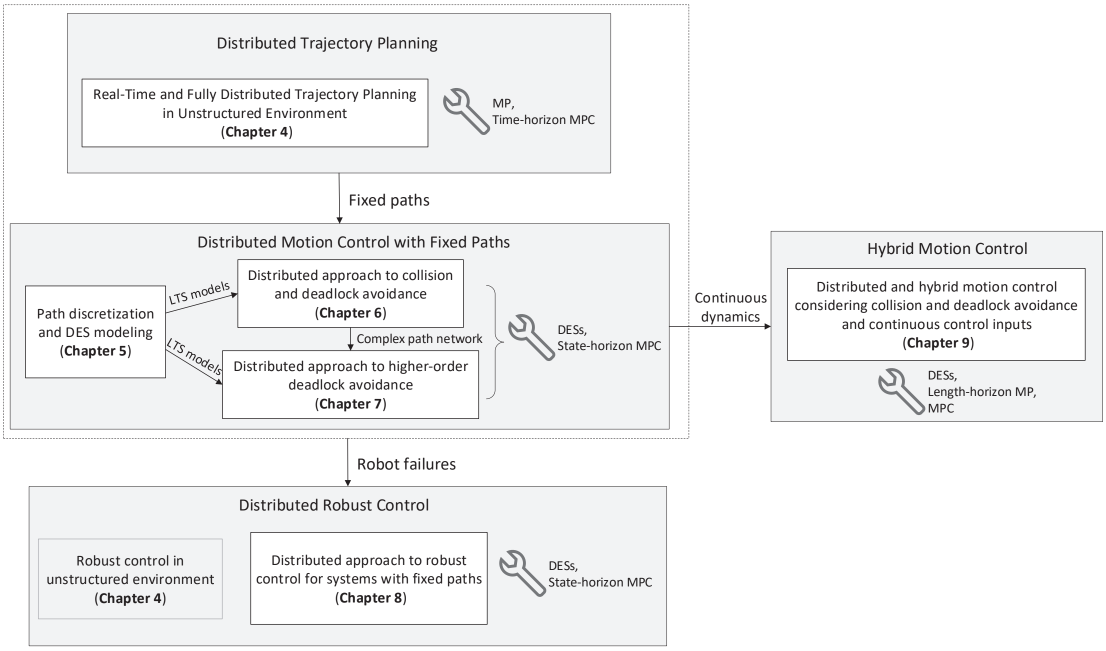


FIG. 1.3: Research contribution of the thesis.

finish its motion tasks, we may further require it to finish tasks with some given objectives, such as optimizing motion smoothness and stability, and maximizing permissive motion. This is the fourth level, i.e., performance optimization, and it is optional.

To achieve the above requirements during robot motion, discrete methods and continuous methods are widely used. Discrete methods usually partition the collision-free environment into a set of discrete states, and the motion of robots can be modeled explicitly or implicitly by discrete event systems, such as Petri nets and transition systems. Based on supervisory control theory, discrete methods can deal with collisions and deadlocks efficiently. Continuous methods study motion control by considering the physical constraints of robots, such as kinematic/dynamic equations and acceleration and velocity limitations. This kind of methods can deal with the physical limitations of robots, and the outputs can easily feed back to robots. Besides, with the technologies of optimization, they can also obtain required optimal outputs.

To leverage the advantages of discrete and continuous methods, this thesis applies supervisory control of discrete event systems (DESs) and mathematical programming (MP) as the main technologies to investigate distributed approaches to motion planning and control in a multi-robot system. The main work and contributions are given in Fig. 1.3. To realize real-time motion, model predictive control (MPC) is applied in the thesis.

As shown in Fig. 1.3, *our first work focuses on distributed trajectory planning*. Most of the current distributed techniques work only for distributed computation but all robots should move simultaneously. This is because to resolve its local optimization problem, a robot needs the computation results of the previous robots. Hence, a robot cannot move with its new control inputs until all robots finish their computation. Our work proposes a fully distributed method for robots moving in an unstructured environment. With the time-horizon MPC strategy, on its current horizon, a robot communicates with its neighbors to obtain their current states, which can be gotten immediately. Then it predicts its neighbors' positions on the current horizon and builds its local optimization problem. The local optimization problem is then resolved independently using sequential convex programming. In this way, each robot can compute and move in a fully distributed way and there is no synchronization of time discretization or prediction horizon. The technologies applied in this work are time-horizon MPC and MP. This work answers our first research question.

When paths are obtained from trajectory planning or path planning, the future robots sometimes are fixed to move along these paths due to similar tasks or infrastructure limitations. For example, in transportation systems, the routes for UGVs are fixed. Hence, in the sequel, *we study distributed approaches to motion control in a multi-robot system where each robot has a fixed path*. We first propose a distributed method to partition paths into collision and collision-free segments. Each robot uses its sensors to detect other paths that intersect with its path and determines the maximal continuous segments whose distances to others are less than the given safe radius. After the discretization of its path, a robot needs to communicate with others so to abstract its collision states. According on the abstracted states, a robot models its motion as a labeled transition system (LTS). Based on the LTS models, we study a distributed approach to avoiding collisions and deadlocks. On each horizon, a robot needs to determine whether its current move transition can be fired or not. The robot checks collisions by directly monitoring the status of its next state. To predict deadlocks, it needs to communicate with others via a multi-hop communication path. Moreover, for some complex path networks, to avoid a deadlock may cause another "circular wait", and recursively cause more "circular waits". We call this situation higher-order deadlocks, which are current deadlock-free but will inevitably lead to deadlocks with the evolution of the system. We introduce the concept of deadlock orders and propose a distributed approach to avoiding higher-order deadlocks. Communications are required to check higher-order deadlocks for a robot. The technologies applied to avoid collisions and deadlocks are state-horizon MPC and supervisory control of DESs. The second question is resolved in this part.

In practice, we cannot guarantee that all robots can work well. Instead, a robot may fail unexpectedly. Hence, by labeling robots with reliable and unreliable, *the third part of the thesis focuses on robust control for a multi-robot system such that the failures of robots have the least adverse effects on the whole system, i.e., block the minimal number of robots*. For systems where each robot can replan its path, robust control can be achieved easily by regarding the failed robots as obstacles. For systems with fixed paths, based on the LTS models, we propose a distributed approach to robust control. On each horizon, a reliable robot checks whether there are any unreliable robots at its current continuous sequence of collision states, and an unreliable robot should communicate with others to determine whether it will move into their current continuous sequences of collision states. In the proposed approach, state-horizon MPC and supervisory control of DESs are applied. This work focuses on the third research question.

As described before, DESs based methods cannot deal with continuous dynamics of robots or generate control input acting on robots' actuators directly. To deal with both deadlock avoidance and robots' kinematics, *we propose a distributed and hybrid method to generate collision-free and deadlock-free continuous control inputs*. At each discrete state, the related path segment is divided into a set of equal-length subsegments. Using length-horizon MPC, each robot on each horizon performs two stages to generate its control inputs. At the first stage, to avoid collisions and deadlocks, discrete control determines the robots, if any, that it needs to wait for via a set of communications. At the second stage, continuous control predicts its waiting time and builds a local optimization problem taking into consideration the kinematic equations and time constraint of the robot. Solving the optimization problem then generates the control input for the motion along the current subsegment. Once the robot reaches the next subsegment, a new horizon begins. This approach depends on the technologies of length-horizon MPC, supervisory control of DESs, and MP. This work is also related the second question, integrating the technologies developed in our first work.

1.3 Contributions of the Thesis

The main contributions of this thesis are highlighted in the following aspects.

First, we propose a real-time and fully distributed algorithm for trajectory planning in multi-robot systems moving in unstructured environments. With the current states of its neighbors, a robot can not only generate its trajectory in a distributed way but also perform its motion in a distributed way.

Second, we propose a distributed algorithm for collision and deadlock avoidance in multi-robot systems with fixed paths. A distributed method to predict deadlocks is described in this algorithm.

Third, we introduce the concepts of higher-order deadlocks and their orders. To the best of our knowledge, they are the first ones in literature. A distributed approach to avoiding higher-order deadlocks is proposed.

Fourth, we study robust control in multi-robot systems, which aims to minimize the number of blocked robots in a system. We propose two distributed algorithms, one for reliable robots and one for unreliable ones, for robust control in multi-robot systems with given path networks.

Fifth, we propose a hybrid and distributed approach to motion control in multi-robot systems with fixed paths. It can not only deal with collisions and deadlocks efficiently, but also generate continuous inputs for the actuators of robots.

At last, our work is an expansion of the supervisory control theory of DESs. We not only apply the supervisory control theory of DESs for motion planning and control in multi-robot systems, but also expand the idea of MPC strategy to DESs and propose state-horizon MPC and length-horizon MPC.

1.4 List of Materials Related to the Thesis

The thesis mainly contains the materials from the following papers.

1. **Yuan Zhou**, Hesuan Hu, Yang Liu, Shang-Wei Lin, Zuohua Ding. "A real-time and fully distributed approach to motion planning for multirobot systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Oct. 2017. http://ieeexplore.ieee.org/document/8055437/.

2. **Yuan Zhou**, Hesuan Hu, Yang Liu, and Zuohua Ding. "Collision and deadlock avoidance in multirobot systems: A distributed approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 7, pp. 1712–1726, Jul. 2017.

3. **Yuan Zhou**, Hesuan Hu, Yang Liu, Shang-Wei Lin, and Zuohua Ding. "A distributed method to avoid higher-order deadlocks in multi-robot systems," *Automatica*, 2018. Submitted.

4. **Yuan Zhou**, Hesuan Hu, Yang Liu, Shang-Wei Lin, and Zuohua Ding. "A distributed approach to robust control of multi-robot systems," *Automatica*, vol. 98, pp. 1–13, Dec. 2018.

5. **Yuan Zhou**, Kun Cheng, Hesuan Hu, Shang-Wei Lin, Yang Liu, and Zuohua Ding. "A hybrid approach to distributed motion control for multi-robot systems," 2019.

1.5 Outline of the Thesis

The rest of this thesis is organized as follows.

Chapter 2 summarizes the state-of-the-art technologies for motion planning, deadlock avoidance, and robust control.

Chapter 3 prepares some basic preliminaries for this thesis, including illustrations of multi-robot systems and LTSs, and the basic procedures of MPC and SCP.

Chapter 4, from Paper 1, gives our first work on real-time and fully distributed approach to trajectory planning for multi-robot systems working in an unstructured environment. This chapter contains the problem statement, construction of local optimization problem for a robot on each horizon and its SCP-based solution, simulation experiments, discussion, and conclusion.

Chapters 5 – 7, from Papers 2 and 3, describe our work on distributed approaches to collision and deadlock avoidance in multi-robot systems where each robot has a predefined path. Detailedly, Chapter 5 builds LTS models to describe robot motion in the system, including collision segments detection, discrete state abstraction, and the resulting LTS models. Chapter 6 proposes a distributed and real-time approach to avoiding collisions and deadlocks based on the LTS models. We first study collisions and deadlocks among robots in terms of LTSs, following what we propose a distributed approach to avoiding collisions and deadlocks. Chapter 7 further investigates higher-order deadlocks and their distributed avoidance. We first study the structural properties of higher-order deadlocks from the system-level perspective, and then a distributed approach is proposed to avoid higher-order deadlocks.

Chapter 8, from Paper 4, focuses on robust control in a multi-robot system where each robot has a fixed path. We first describe our control target in terms of LTS models, and then propose two distributed algorithms, one for reliable robots and the other for unreliable ones, to achieve robust control.

Chapter 9, from Paper 5, studies a distributed and hybrid approach to motion control in the system described in Chapter 5. We first describe the discrete part, which detects the robots that a robot needs to wait for at its current state in order to avoid collisions and deadlocks. Then we study the continuous part, which builds a local optimization problem considering the possible waiting time and resolves it to produce the current acceleration. The communication protocols are given in the paradigm of Petri nets.

Chapter 10 finally concludes the thesis and provides an outlook on the future research directions based on the current work.

Chapter 2

Related Work

Motion planning and control for robots is one of the most important and active topics in robotics. In this chapter, we give a comprehensive literature review on the state-of-the-art methods of motion planning and control for robots to avoid collisions and deadlocks, and guarantee robustness.

2.1 Motion Planning

During the last decades, motion planning have been widely studied by researchers [12–17]. The task of motion planning is to generate a feasible, even optimal, trajectory or path for a robot such that the robot can move from its initial position to its target without causing collisions. Many methods have been developed based on some typical and attractive technologies: formal methods, state lattice, cell decomposition, sampling, roadmap, bug algorithms, potential fields, velocity obstacles, mathematical programming, spline curves, and reinforcement learning.

Formal methods [18–28] apply the technologies such as verification and model checking [29–31] to control robots' motion. The motion is first modeled by such as automata and Petri nets, while the requirements, e.g., collision avoidance, are modeled by logical descriptions, such as LTL (linear temporal logic) and CTL (computation tree logic). Thus, model checking technologies can be applied to determine a sequence of transitions or actions that satisfies the given LTL or CTL specifications. For example, Saha *et al* [22] propose a compositional framework for motion planning of multi-robot systems. The possible motion of a system is modeled as a special transition system based on the given library of motion primitives, each of which leads to a transition of the system. The safety and behavioral properties are described by LTL specifications. An satisfiability modulo theories (SMT) solver then can automatically generate robots' trajectories, which are characterized as the compositions of motion primitives.

Roadmap-based methods are widely studied in literature, such as [12, 32–46]. A road map is a graph in the collision-free space where a vertex is a collision-free configuration, and an edge represents a unique collision-free path, e.g., line segments, between its two endpoints in the collision-free space. Once a road map is generated, graph searching algorithms, e.g., A* algorithm, D* algorithm, and fast marching method, can be used to generate a path connecting the initial and target positions. The main process for this kind of methods is to build the road map in the collision-free configuration. Some common technologies to efficiently build a road map are visibility graph methods [36–39], Voronoi diagrams [40–43], and sampling methods [44–46]. The initial idea of visibility graph is developed for polygonal obstacles, where the vertexes of obstacles are the nodes of the graph, and two nodes are connected if the line segment joining them does not pass through any obstacles. Some improvements focus on the refining of the graph and approximation for non-polygonal obstacles. Voronoi diagrams are based on Voronoi regions with respect to obstacles. Based on the obstacles, the configuration is partitioned into a set of Voronoi regions, and the boundaries of these regions form a Voronoi diagram, which is the generated road map. The above two technologies are complete, but the computation efforts may be large. Probabilistic roadmap (PRM) in an alternative method to generate a road map with low computation cost. The idea is to generate a set of sampling points, including the initial and target points, in the collision-free configuration space, and two nearby points are connected if the path, generated by simple and fast local planners (e.g., linear motion), is feasible. Note that PRM is provably probabilistically complete and the rate of convergence depends on certain visibility properties of the free space. Recently, for rural environment navigation [47], a road map was first roughly generated based on the "topological" map of the environment, which may be not exactly precise but at least the outline of a path can be obtained; the real motion along each edge is then refined by the local perception system.

Cell decomposition methods [12, 48–52] concentrate on the partition of the configuration. In cell decomposition methods, the collision-free configuration space is first partitioned into a set of adjacent but disjoint cells, either regular (approximate cell decomposition) or irregular (exact cell decomposition) [12]. A robot can only move to an adjacent cell from its current cell. Then any search algorithms, such as A* algorithm or D* algorithm, can be used to determine a sequence of cells that the robot needs to pass through. Two main problems in cell decomposition based methods are the construction of cells [12, 52] and the motion within cells [48, 50]. For example, Dugarjav *et al* in [52] propose an online cell decomposition method to explore an unknown environment in real time. Based on the detected environment with sensors, each cell is composed and updated until it is unchanged. The work in [48, 50] applies roadmap based method to navigate robot motion in each cell.

The general idea of state lattices [53–57] is to build a state lattice. A state lattice is a connected graph where the vertexes, denoting the discrete states that a robot can arrive at, are connected by specific patterns, representing the set of elementary motions or motion primitives. Detailedly, the configuration space of a robot is first sampled regularly by a set of grid points; then a finite set of feasible motion primitives are generated to connect the discrete points. Hence, a state lattice is built. Similarly, any graph search methods can be applied to search for the state lattice to generate a route from the initial state to the target one.

Sampling-based methods [58–71] are another kind of the state-of-the-art motion planning technologies. Rather than explicitly exploring the obstacles, sampling-based methods regard the collision detection as a "black box" and probe the configuration space with a sampling scheme [14]. Two popular kinds of sampling-based methods are the PRM methods [44–46] and rapidly-exploring random trees (RRTs) methods [67, 69, 70]. The former one is a one-round sampling scheme, while the latter one is an incremental sampling scheme. As described in the roadmap based methods, given the initial and target positions, PRM methods generate all samples and build a road map from the initial position to the target one. However, rather than construct the road map in advance, RRT methods execute a single-query scheme and build a tree incrementally from the start configuration to the goal configuration, or vice versa. At each round, it first generates a sample in the collision-free configuration and selects the nearest vertex for the sample in the current tree; then generates a new configuration with a given distance to the nearest vertex along the direction to the sample one; finally adds an edge to the nearest configuration to the new configuration. Similarly, once a tree is generated, a graph search algorithm generates a path.

All the above technologies are discrete methods, which discretize the configuration space into a set of discrete states, and then motion planning is to select a sequence of discrete states that a robot needs to pass through. Discrete representation of a robot's motion can reduce computation complexity. However, it sacrifices feasibility, i.e., computing a motion that satisfies different constraints, and optimality, i.e., generating a motion that satisfies some optimal objectives. Hence, due to its capability to describe robots' physical dynamics and generate continuous values of motion variables (e.g., velocities or accelerations), continuous methods are also widely studied in robotics.

Bug algorithms [13, 72] are the simplest algorithms and can be implemented easily. The basic idea of Bug algorithm is to control each robot move directly to its target. Once an obstacle is found, the robot follows the boundary of the obstacle until the latter is passed through, and then the robot moves to its target again. Some variants focus on the determination of positions that a robot can move directly to its target. Due to its simplicity, the generated path usually is not optimal.

Potential fields-based methods [73–83] are another kind of well-designed continuous methods with physical metaphors. The main step in potential fields is to define proper attractive potential functions, which can lead a robot to its target, and repulsive potential functions, which drive robots away from the obstacles. Such functions can be designed on the basis of forces, accelerations, and velocities. However, general potential fields are likely to cause local minima, which may cause robots not arrive at their targets. As an improvement, navigation functions are proposed to deal with local minima. This kind of technologies builds a potential in a transformed space and maps it back to the configuration space. For example, in [75], the configuration space is mapped to a unit disk and then artificial potential field augmented with an appropriate adaptive control law is applied to generate the velocity of a robot. However, navigation functions require complete knowledge of the environment and thus are off-line. Xu *et al* propose moment-based methods [84, 85], where the repulsive moments of obstacles and the attractive moments of targets are designed to guarantee robots to avoid collisions and converge to targets.

The methods based on velocity obstacles [86–88] focus on the velocity space and are proposed initially for a robot with moving obstacles. Subsequently, reciprocal collision avoidance based on the concept of velocity obstacles [89, 90] and its variations, such as optimal reciprocal collision avoidance [91, 92], acceleration velocity obstacles [93], and others [94, 95], are proposed for multiple robots. A velocity obstacle is the set of velocities of a robot that will result in a collision with a moving obstacle at some moment in the future. In velocity obstacle-based methods, the robot is the only one that takes the responsibility to avoid collisions with obstacles, while in reciprocal collision avoidance and its variations, considering reciprocity, each robot takes the responsibility to avoid collisions. For example, in optimal reciprocal collision avoidance, the set of safe velocities is evenly divided between two robots by defining halfplanes of safe and possible velocities, which are the sets of individual velocities for two robots that result in relative velocities outside of the velocity obstacle. The main assumption for this kind of methods is that the velocity of a robot keeps constant over a finite time interval.

Mathematical programming is one of the most powerful technologies for motion planning and control [96–106]. The key task for this kind of methods is to model the motion planning problem as a proper optimization problem, such as mixed-integer optimization problem, quadratic optimization problem, convex optimization problem, and so on. Since we can add different constraints, such as kinematics, dynamics, communication connectivity, target tracking, and collision avoidance, to build the optimization problem, mathematical programming-based methods have great capability of application. Combining with the MPC strategy, we can also apply it to do real-time planning [103–106]. There are usually two ways to construct optimization problems. The first one is that for each multi-robot system, a coupled optimization problem is built and solved, which will generate the control inputs of all robots in the system, such as the work in [97]. This kind of methods can obtain the maximal motion performance of robots, however, the problem usually is very large and the computation cost is high. Another one is to build a set of local optimization problems, each of which is assigned to a robot, such as the work in [103]. Such decoupling can reduce the scale of the problem and computation cost, but sacrifices some degrees of motion performance.

Spline curve based methods [107–111] plan robots' paths via predefined spline curves, such as β-curve, Bézier curves, and Bernstein curves. The priori assumption is the path of a robot is composed of a set of piecewise predetermined spline curves. Hence, the planner needs to determine the parameters of these spline curves so as to generate a smooth path or other objectives.

Recently, with the development of machine learning technologies, reinforcement learning has been a popular technology for robot motion planning and control in both discrete and continuous motion spaces [112–118]. By modeling the motion of a robot as a Markov decision process, reinforcement learning plans an action policy to reach a desired goal state, through the maximization of a value function. With different forms of Markov decision models, it can be either a discrete or a continuous method. Under a reinforcement learning planner, at each discrete time instant, a robot chooses an action from the set of available actions to maximize the value function based on the current observation. With the selected action, the robot can move to the next state. The primary advantage of reinforcement learning lies in its inherent power of automatic learning even in the presence of small changes in the world map. Different reinforcement learning algorithms, such as Q-learning and TD(λ), have been proposed [119].

The motion planning approaches we have described so far are either discrete or continuous. There are also some works on the hybrid methods combining discrete and continuous methods [120–127]. However, most of the current hybrid approaches focus on both task allocation and motion planning. The discrete parts are for switching among different tasks and the continuous parts are to generate continuous motion to finish the related tasks. For example, Guo *et al* [123] proposed a hybrid approach to a team of robots moving with contingent temporal tasks and formation constraints. The motion of each robot is divided into two modes: navigation control and formation control. For each mode, a navigation function method is designed to control robot motion.

All the above methods are suitable for robots moving in an unstructured environment where robots can replan their paths or trajectories freely. However, sometimes, due to infrastructure limitations (e.g., intelligent transportation systems and warehouse), or priori path planning (such as the work in [53]), or previous robots' motion, robots are fixed on predetermined paths. For these cases, motion control for collision avoidance is achieved by controlling robots to traverse a collision location at different times [128–132]. For example, Smith and Rus *et al* [130] studied collision avoidance by repeatedly resuming and stopping robot motion based on some stopping strategies such that they can pass through the collision zones at different times. Wang *et al* [132] propose a method to avoid collisions by assigning different initial time delays to robots.

Motion planning determines the reference paths or trajectories of robots. However, in practice, a robot may not move along the predefined path or trajectory exactly. To guarantee that a robot can track its reference path or trajectory as accurate as possible, researchers have designed some special tracking controllers, such as pure pursuit-based controllers [133–135], PID (proportional, integral, and derivative) controllers [136–138], fuzzy logic controllers [139, 140], and mathematical programming based controllers [141–143], sliding mode control [144]. For example, the idea of pure pursuit algorithms is that given a path and the nearest point on it with a given lookahead distance from the current position, the real tracked path between the current position and this position is a curve with a constant curvature, based on which the robot can compute a steering command for the motion direction. A PID controller computes an error value continuously and determines the control function based on three terms: proportional (proportional to the current error), integral (integral of past errors), and derivative (change rate of the current error).

Table 2.1 gives a brief summary of the state-of-the-art motion planning methods. Even though motion planning has been widely studied, there are some open problems that are not adequately addressed. First, most of the current approaches are either centralized or decentralized. Centralized methods lack the flexibility and robustness, while decentralized ones may lead to low performance. Even though there are some distributed approaches recently proposed, most of them focus on distributed computation, but

TABLE 2.1: Summary of Different Motion Planning Algorithms

	Methods	Key Steps	Representative Literature
Discrete Methods	Formal Methods	Describe requirements using LTL and/or CTL	[18, 19, 22]
	Roadmap methods	Build a road map (visibility graph, Voronoi diagrams, sampling methods) and perform graph search algorithms	[38, 41, 45]
	Cell decomposition methods	Partition the configuration space into a set of adjacent cells and perform graph search algorithms	[12, 49, 50]
	State Lattices	Build a state lattice based on motion primitives and perform graph search algorithms	[54, 55]
	Sampling methods	Generate a set of samples and check collision avoidance between any two samples. Two main methods are PRM and RRTs	[45, 67]
Continuous Methods	Bug Algorithms	Move directly to the target and follow the boundary of the obstacles	[13, 72]
	Potential Fields	Build attractive and repulsive potential functions	[73, 76, 77]
	Velocity obstacles	Construct a proper velocity obstacle, i.e., the set of velocities of a robot that will result in a collision, in the velocity space	[87, 89, 91]
	Mathematical Programming	Construct a proper optimization problem and solve it efficiently	[96–98]
	Spline Curves	Select predefined spline curves and determine proper parameters of these curves	[108–110]
Others	Reinforcement Learning	Training data collection and model determination	[115–117]
	Time Control	Compute different time delays such that different robots pass through the same position at different times	[130, 131]

all robots should move synchronously rather than distributively. Second, given a motion task, most of the current approaches are either based on discrete abstraction or on continuous models. Discrete abstraction usually cannot obtain the low-level inputs for actuators directly; while continuous methods may cause high computation complexity.

2.2 Deadlock Avoidance

During the motion of multiple robots, deadlocks may occur among robots in order to avoid collisions. The occurrence of deadlocks is also dangerous for a multi-robot system since deadlocks will stop robots from moving forward and even stagnate the whole system, which will degrade system performance inevitably. Indeed, deadlock avoidance is a great challenge to systems containing multiple subsystems with shared resources, such as automated manufacturing systems and multi-robot systems. Since the four conditions, which are effectively defines a deadlock, were proposed in 1971 [145], deadlocks have been widely studied in automated manufacturing systems and multi-robot systems, such as [146–170] and the references therein. Note that these two kinds of systems are almost similar. Indeed, in a multi-robot system, the configuration space can be regarded as the set of resources and each robot's motion is a process. Among the existing works, there are mainly three strategies to solve deadlocks in systems: deadlock prevention [147–155], deadlock recovery [156, 157], and deadlock avoidance [158–170].

Deadlock prevention is an off-line mechanism to avoid deadlocks. The main step for deadlock prevention is to compute liveness conditions or design a proper controller before a system is released such that deadlocks can never occur [147–152]. For example, for Petri net models, we can apply state-based methods, e.g., finding conditions of markings which guarantee system liveness [147], or structure-based methods, e.g., designing control policies guaranteeing that the siphons are non-empty [148]. However, such strategies are with exponential computation complexity with regard to the size of the nets [149]. Currently, some work focusing on decentralized control with local search is also proposed [151, 152].

For deadlock recovery methods, deadlocks are resolved once they are detected [146, 156, 157]. The main characteristic of this kind of strategies is that it allows the occurrence of deadlocks since there are no checks before the execution of processes. Once a deadlock is detected, some resolution methods are applied to resolve the deadlocks. For example, in [157], deadlocks among multiple mobile robots are detected by dynamically constructing and searching a waiting graph, where each node corresponds to a robot and a directed edge between two nodes indicates that one robot is waiting for another. Deadlocks are resolved by changing edge directions, i.e., changing motion directions, or node connections, i.e., replanning motion trajectories, to avoid cycles in the waiting graph. However, because of the existence of deadlocks, it is suitable for the systems where deadlocks are rare and cannot result in severe catastrophes, and the recovery is affordable [146].

Deadlock avoidance is an online strategy to avoid deadlocks. Via looking ahead into the future system evolution, it first predict online whether the current evolution would cause deadlocks and then take necessary actions to avoid deadlocks. Thus, no deadlocks can occur during the evolution of a system [158–170]. Centralized or decentralized methods are used to predict deadlocks. Centralized methods usually focus on structure analysis or reachability space of the whole system. For example, Yalcin *et al* [165] use finite automata to model the manufacturing cells and the process plans. Based on these automata, deadlocks are predicted and avoided by analyzing the state space. Centralized methods can obtain high efficiency but cause high cost because of the building and searching of the state space. While for decentralized methods, each process or robot predicts its local evolution and checks whether there may cause deadlocks. For example, Lee *et al* [161] study deadlock avoidance in zone-control automated guided vehicle systems. After modeling the system via Petri nets, each vehicle predicts deadlocks by checking the results of firing of its remaining transitions. The main cost for this strategy is the prediction of deadlocks. Since the optimal deadlock avoidance is NP-complete [171], some conservative methods are proposed in practice, such as the Banker's algorithm [172] and its variations [162, 163]. The Banker's algorithm requires that at any time, the move of a robot should guarantee that each robot can move to its destination sequentially in some order. While its variations focus on more relaxed movable conditions. For example, in [162, 163], a robot can move forward if it can move to a location that cannot be occupied by others, rather than to its destination. Decentralized methods can predict deadlocks with lower computation cost but may prevent many admissible motions.

Three main tools used in above strategies are graph theory, automata, and Petri nets.

Digraphs are an intuitive instrument to detect and avoid deadlocks. The main idea is to use digraphs to model the request-supply relations between processes and resources in a system, and then detect and avoid deadlocks by searching and avoiding cycles in the graphs, such as the work in [130, 156–158]. For example, in [130], deadlocks are detected and resolved in real time based on deadlock graph, a directed graph where an edge from node i to node j means that robot i is stopped waiting for robot j. At each time instant, if a cycle is to exist in the built deadlock graph, then deadlock is avoided by deleting one of the edges in the cycle, meaning that one of the robots resumes its motion.

Automata are another efficient tool to solve deadlock problems, such as the work in [153, 154], and [165]. As stated in [153], with the supervisory-control theory developed by Ramadge and Wonham (R-W theory), automata theory would directly yield deadlock-free supervisors during the construction of an automaton. Such supervisors determine formally the set of states could reach as well as the set of events allowed to occur at those states. With this statement, the authors in [153] study deadlock-free schedules using time-augmented automata and A* algorithm.

Petri nets are also a powerful instrument to model the system dynamics and resolve deadlocks. One can usually take advantage of the reachability graph, siphons, and liveness of Petri nets to characterize deadlocks, such as the work in [146–152, 159–161]. Petri nets can be used either for deadlock prevention or deadlock detection and avoidance. For deadlock prevention, supervisory controllers are designed in advance based on structural analysis, i.e., siphons, such that no deadlocks occur, or proper initial markings are designed based on reachable state analysis to avoid deadlocks. For deadlock detection and avoidance, Hu *et al* propose a set of decentralized deadlock avoidance algorithms, e.g., [159, 160], by predicting whether a process can safely reach to a place without sharing any resources based on the current available resources.

Table 2.2 gives the summary of different deadlock resolution strategies. Due to real-time change of the environment, we focus on deadlock avoidance in multi-robot systems. Except the fruitful results on deadlock avoidance, the balance between computation complexity and system performance is still a great challenge. Centralized methods can obtain the highest performance but is with a high computation complexity, while decentralized methods reduce computation complexity significantly at the expense of performance. Much attention is still paid on how to achieve a good performance with acceptable computation cost.

TABLE 2.2: Summary of Different Strategies for Deadlock Resolution

Deadlock Resolution	Main Idea	Key Characteristics	Representative Literature
Deadlock Prevention	Design a deadlock-free controller during the design of a system	Off-line; High complexity	[147, 148]
Deadlock Recovery	Change the behavior of a subsystem once a deadlock is detected	Allow the occurrence of deadlocks	[156, 157]
Deadlock Avoidance	Predict deadlocks in advance and then take actions to avoid them	On-line; Centralized or decentralized	[161, 162, 166]

2.3 Robust Motion

Robust control of the robotic systems is also widely studied, such as [173–189] and the references therein. These methods can be roughly divided into three categories.

The first one is to obtain robustness by giving the system some degree of redundancy, so that the tasks can still be completed by others even when some robots fail unexpectedly, e.g., the work in [173–177]. For example, Dias *et al* [173] study the means to ensure the robustness in a robot team when malfunctions occur. A set of redundant strategies are proposed such that the tasks bestowed to the failed robots can still be finished by other correctly-running robots. Thus, the team can still complete the given tasks even when some robots fail. In [174], collaborative control, i.e., multiple sources share the control of a single robot, is used to guarantee the robot's motion robustness against the malfunctions of some resources. The main challenge of this kind of methods is to select proper numbers of spare components or robots since a full backup is consuming.

The second one is to add some mechanisms, which are used to detect failures so as to recover/reconfigure the robots, into the system, such as the work in [178–182]. For example, Dogar *et al* [178] propose a hierarchical planning approach to accomplishing some multi-scale assembly operations. The robustness is achieved by the process of failure detection and recovery: Once a scanner loses the track of a target object, the system reverts back to an earlier stage in order to re-localize by using a wider field of view systems. Hofbaur *et al* [179] propose a generalized framework to improve the robustness of the motion of mobile robots. The proposed framework can automatically monitor the driving device of a mobile robot and reconfigure the robot in cases of failures. Thus, high-level control like path-planner is only to change its behavior in case of a serious damage. However, some failures are hard to detect; some may take a long time to fix; some cannot be recovered on-line.

The last one is based on relaxing requirements or motion. In order to obtain the robustness against uncertainties or disturbance of robots and environment, the system is designed to be endowed with additional flexibility in terms of either deterministic or probabilistic models, such as the work in [183–189]. For example, Blackmore *et al* [184] use a probabilistic approach to planning vehicles' flexible trajectories. Each trajectory is described by the probabilistic distribution of a vehicle's states. The probabilities of collisions along these trajectories are designed to be below a given threshold. Thus, each vehicle has the ability to deal with uncertainties, such as indefinite localizations, erroneous modelings, and unexpected disturbances. Hence, the whole system can execute robustly. Liemhetcharat and Veloso [187] study the method to select a team of robots, each of which has a failure probability, to construct a robust system. The robustness they consider is the probability of the performance exceeding a threshold. The algorithms they propose are to maximize the robustness of the system. Sun *et al* [182] study robust control of robots' motion by rendering the real trajectory in a tube centered along the reference one.

Most of the current work regarding robust control focuses on the system's capability to tolerate failures, changes, and disturbance so that the system can still work well or complete its tasks. Sometimes, we cannot guarantee that robots will not fail or the system can always complete its tasks. Once a robot fails inevitably and the system cannot complete its tasks anymore, how to minimize the detrimental efforts of a failed robot on other robots and maximize the performance of the system becomes more important. However, there is a little work focusing on this topic in automated manufacturing systems, such as [190–192].

Chapter 3

Preliminaries

In this chapter, we describe some terminologies and preliminaries used throughout this thesis.

3.1 Multi-Robot Systems

A multi-robot system is a system that contains multiple robots moving in a given environment. Suppose the workspace of the system is $\mathscr{W}$, where $\mathscr{W} \subset \mathbb{R}^{n_0}$ and $\mathbb{R}^{n_0}$ is the n_0-d Euclidian space. This means each robot can only move in $\mathscr{W}$. Note that the workspace $\mathscr{W}$ may contain obstacles $\mathcal{O}$. In the sequel, we give some basic definitions and assumptions of a multi-robot system used in this thesis.

Definition 1 (Path). Given the motion space $\mathscr{W}$, a path of a robot, denoted as p, from its initial position $\boldsymbol{x}_0 \in \mathscr{W}$ to the target $\boldsymbol{x}_f \in \mathscr{W}$, is a geometric curve in $\mathscr{W}$, which is defined by a parameter equation: $p = p(\theta)$, $\theta \in [0, 1]$, mapping from $[0, 1]$ to $\mathscr{W}$, where $p(0) = \boldsymbol{x}_0$ and $p(1) = \boldsymbol{x}_f$.

A path of a robot is independent of time; it describes the sequence of positions that a robot needs to move to, but it does not stipulate the time that a robot needs to arrive at each point.

Definition 2 (State). The set of attribute values identifying the status of a robot during its motion is called a *state*, denoted as s. The set of all states is called *state space* of the robot, denoted as S.

For example, in some discrete methods, robot motion is usually expressed as a formal model, e.g., transition systems or Petri nets; in these cases, a state of a robot is a state reachable in the formal model. In continuous methods considering the kinematics of a robot, the status of a robot is usually characterized by the set of position, velocity and acceleration; hence, a state of a robot is the value vector of position, velocity and acceleration.

Definition 3 (Trajectory). A trajectory of a robot r_i, denoted as q, from the initial state q_0 to the target q_f, is a time parameterized function: $q = q(t)$, $t \in [0, \tau]$, mapping from the time interval $[0, \tau]$ to its state space S, where $q(0) = q_0$ and $q(\tau) = q_f$.

Different from a path, a trajectory of a robot is parameterized by time and describes where and how a robot moves during its motion.

Definition 4 (Configuration). Given a multi-robot system with N robots $\{r_i, i = 1, 2, \ldots, N\}$, the status of the system is called a *configuration*, denoted as c, which is the set of the states of all the robots, i.e., $c = (s^1, s^2, \ldots, s^N)$, where $s^i \in S^i$ is the state of r_i and S^i is the state space of r_i.

Basic Assumptions. The evolution of a multi-robot system relies on a lot of things, such as the motion control algorithms, the sensors to monitor the environment, the communication via wireless network, and so on. However, we cannot deal with all of them in this thesis. As usual, the clarity of one perspective's discussion can be attained by the negligence of others, i.e., their correctness is assured by default. In this thesis, we focus on the design of planners for motion planning and control. Thus, to simplify the problem, we need some additional assumptions. Note that if an assumption is not satisfied, we can refer to solutions in the related community to fix it first.

1. Location and Communication Assumptions. There are two kinds of ranges for each robot. One is sensing range, and the other is communication range. The sensing range relies on the sensors to be deployed, such as laser sensors; while the communication range is based on the wireless network. Usually, these two kinds of ranges

are mutually independent. However, communication range should be larger than sensing range since a robot needs to communicate with the robots within the sensing range for the sake of collision avoidance. Moreover, we assume that each robot can locate other robots or obstacles within its sensing range using the sensors.

2. Each robot can communicate with its neighboring robots within the communication range directly. Via a multi-hop communication path, a robot can further communicate with the robots beyond the communication range. We do not consider packet delays, errors, and drops during robots' communication.

3. Robot Assumptions. With proper actuators, a robot can always move along the desired path with a tolerable derivation. This derivation can be addressed by constraining the robot into the safe radius.

3.2 Labeled Transition Systems

In literature, there are many formal models applied to model robot motion, such as automata, Petri nets, and transition systems [29]. In this thesis, labeled transition systems are applied due to their generic semantic and wide applications.

Definition 5. A labeled transition system (LTS) is a quadruple $\langle S, \Sigma, \rightarrow \rangle$, where

- S is a finite set of states,
- Σ is the set of labels (or actions), and
- $\rightarrow \subset S \times \Sigma \times S$ is a finite set of transitions.

The transition triggered by an event δ from s_i to s_j, i.e., $(s_i, \delta, s_j) \in \rightarrow$, is denoted as $s_i \xrightarrow{\delta} s_j$. Let $Pos(s)$ be the set of succeeding states of s, i.e., $Pos(s) = \{s' \in S : \exists\delta \in \Sigma, \ni s \xrightarrow{\delta} s'\}$. Similarly, the set of preceding states of s is $Pre(s) = \{s' \in S : \exists\delta \in \Sigma, \ni s' \xrightarrow{\delta} s\}$.

For example, Fig. 3.1 shows an example of LTS models. In this example, the set of states is $S = \{s_1, s_2, \ldots, s_7\}$, the set of labels is $\Sigma = \{e_1, e_2, \ldots, e_9\}$, and the

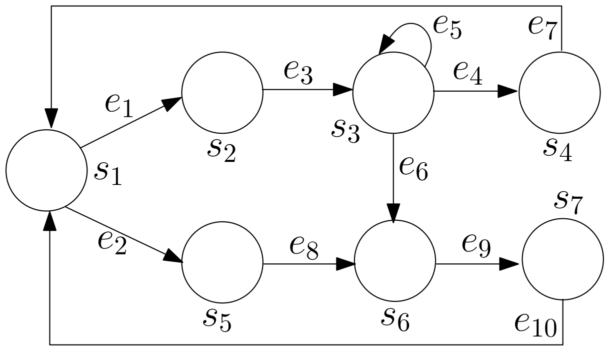


FIG. 3.1: An example of LTS.

transitions are $s_1 \xrightarrow{e_1} s_2$, $s_1 \xrightarrow{e_2} s_5$, $s_2 \xrightarrow{e_3} s_3$, $s_3 \xrightarrow{e_4} s_4$, $s_3 \xrightarrow{e_5} s_3$, $s_3 \xrightarrow{e_6} s_6$, $s_4 \xrightarrow{e_7} s_1$, $s_5 \xrightarrow{e_8} s_6$, $s_6 \xrightarrow{e_9} s_7$, and $s_7 \xrightarrow{e_{10}} s_1$.

3.3 Model Predictive Control

This section gives a brief review of the idea of model predictive control (MPC). Please refer to [193] for details.

MPC is not a specific control algorithm but is a general strategy for a kind of control methods. It applies an explicit model to describe the control system and obtains a sequence of control inputs by solving an optimization problem based on the model. By receding strategy, each time only the first control input in the sequence is applied to the system, and then the horizon moves to the future. The general process of MPC is given in Fig. 3.2. As shown in Fig. 3.2(a), suppose explicit model of the system is $q[t+1] = g(q[t], \boldsymbol{a}[t])$. Given a finite horizon $[k_0, k_0+H]$ from the current time k_0, MPC based methods compute the optimal control signals on the horizon, i.e., $\{\boldsymbol{a}[k_0], \boldsymbol{a}[k_0+1], \ldots, \boldsymbol{a}[k_0+H-1]\}$, as well as the future outputs $\{q[k_0+1], q[k_0+2], \ldots, q[k_0+H]\}$, shown as the dashed curves in Fig. 3.2(a), by solving an optimal problem. Among the optimal inputs on the horizon, only the first signal $\boldsymbol{a}[k_0]$ is adopted as the control input and the process evolves to $q[k_0+1]$, which is shown by the bold line in Fig. 3.2(b). As time proceeds to k_0+1, the horizon recedes to $[k_0+1, k_0+H+1]$ and the related optimal control signals $\boldsymbol{a}[k_0+1], \ldots, \boldsymbol{a}[k_0+H]$ are computed, which are shown in the dashed lines in Fig. 3.2(b).

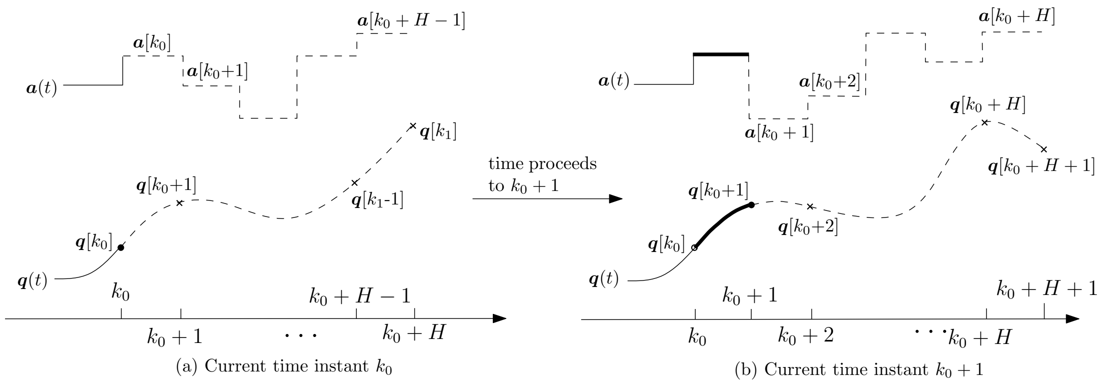


FIG. 3.2: General process for MPC-based control methods.

3.4 Sequential Convex Programming

As described in Section 3.3, an MPC based method needs to solve an optimization problem on each horizon. Usually, this problem is non-convex and is not known to admit polynomial time algorithms. In fact, most are NP-hard such that finding a polynomial time solution is impossible [194]. However, sequential convex programming (SCP) [195] gives a local optimal but efficient approximate method to solve non-convex programming, facilitating the advantages of convex optimization. In this section, we describe some basic knowledge about convex programming and sequential convex programming. For details, please also refer to [196, 197].

Suppose function $f : R^n \to R$ is defined on the domain $D \subset R^n$. f is a *convex function* in D if D is a convex set and $\forall x_1, x_2 \in D$ and $\forall \theta \in [0,1]$, $f(\theta x_1 + (1-\theta)x_2) \leq af(x_1) + (1-a)f(x_2)$. Consider the following general optimization problem:

$$\begin{aligned} \min \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq 0, i = 1, 2, \ldots, m, \\ & h_i(x) = 0, i = 1, 2, \ldots, l. \end{aligned} \tag{3.1}$$

(3.1) is a *convex optimization problem* if $f_0, f_1, \ldots, f_m$ are convex functions, and $h_i(x) = a_i^T x - b_i$, where $a_1, a_2, \ldots, a_l$ are constant vectors, and $b_1, \ldots, b_l$ are scalars.

Next, we give a brief overview of the SCP procedure to solve (3.1) approximately in its general case. The basic idea of SCP is to approximate the original non-convex optimization problem via a sequence of convex optimization problems, whose solutions are convergent to a local optimal solution of the original one. Given an initial value x^0, SCP obtains the approximate solution via iterations. At iteration k with the obtained x^k, $k = 0, 1, 2, \ldots$, it maintains a convex trust region $D(x^k)$ near x^k; then constructs and solves an convex optimization $P_a(x^k)$ of (3.1) over $D(x^k)$; the optimal solution of $P_a(x^k)$ is x^{k+1}, which is used at iteration $k+1$. One possible way to preform the SCP procedure at iteration k can be described as follows. First, the trust region is typically maintained by (3.2), where ρ is a given value.

$$D^k = \{x | \|x - x^k\|_2 \leq \rho\} \tag{3.2}$$

Second, the construction of $P_a(x^k)$ can be done using (3.3)−(3.5).

$$\widetilde{f}_i(x) = f_i(x^k) + \nabla f_i(x^k)^T (x - x^k), \tag{3.3}$$

$$\widetilde{f}_i(x) = f_i(x^k) + \nabla f_i(x^k)^T (x - x^k) + \frac{1}{2}(x - x^k)^T \nabla^2 f_i(x^k)(x - x^k), \tag{3.4}$$

$$\widetilde{h}_i(x) = h_i(x^k) + \nabla h_i(x^k)^T (x - x^k), \tag{3.5}$$

where $\nabla f_i(x^k)$ and $\nabla^2 f_i(x^k)$ are the gradient vector and Hessian matrix of $f_i(x)$ at x^k, respectively. Indeed, for each non-convex function f_i in the inequality constraints, we apply its first-order or second-order Taylor approximation, i.e., (3.3) or (3.4); the affine approximation of each non-convex equality constraint is given by its first-order Taylor approximation, i.e., (3.5); and $\widetilde{f}_i(x) = f_i(x)$ and $\widetilde{h}_i(x) = h_i(x)$ for others. Hence, $P_a(x^k)$ can be described as:

$$\begin{aligned}
\min \quad & \widetilde{f}_0(x) \\
\text{subject to} \quad & \widetilde{f}_i(x) \leq 0, i = 1, 2, \ldots, m, \\
& \widetilde{h}_i(x) = 0, i = 1, 2, \ldots, l, \\
& x \in D(x^k).
\end{aligned} \tag{$P_a(x^k)$}$$

Chapter 4

Fully Distributed Approach to Trajectory Planning for Multi-Robot Systems

In this chapter, we study fully distributed trajectory planning for multi-robot systems, where each robot is equipped with some sensors of limited sensing ranges and moves in an unstructured and changing environment. Fully distributed means each robot will perform both computation and motion in a distributed manner.

4.1 Introduction

In an unstructured environment, there are usually many feasible trajectories in the motion space. One important issue is how to select a trajectory satisfying some requirements, such as shortest moving distance, shortest motion time, or fewest encountered obstacles. In the form of mathematical programming, we can achieve great capability to describe not only multiple constraints simultaneously [96], such as kinematics, dynamics, connectivity, target tracking, and collision avoidance, but also different objectives, such as shortest distance, shortest time, and minimum energy consumption. Hence, mathematical programming is one of the most active technologies.

Almost all the existing mathematical programming-based methods describe motion planning problems in centralized or decentralized forms. For the centralized form, the system is modeled by an optimization programming and the control inputs of all robots are determined simultaneously [12, 14]; while for the decentralized form, each robot is modeled as an optimization programming and robots' decision variables are determined in a sequential manner since the latter one needs some information computed by the previous robots [98, 103]. However, both of them have to control the robots to move simultaneously, and thus the robots lack motion flexibility. 1) Centralized methods can obtain the best cooperation performance of a system, but the computation complexity is very high. Since the control signals of robots are computed at the same time, all robots move simultaneously and the system lacks robustness and scalability. 2) Decentralized methods can reduce computation complexity by decoupling the problem into a set of subproblems, each of which can be solved by an individual robot distributively. However, to solve its own subproblem, a robot may need the computation results from its neighbors. Thus, it needs to explicitly or implicitly assign robots with priority in advance so that they can compute the trajectories sequentially [98, 103]. As a result, robots cannot move forward until all robots finish their prediction.

In this chapter, we propose a real-time and fully distributed trajectory planning method for multi-robot systems. Robots in the system are required to move from the initial positions to the given destinations without collisions. Each robot is equipped with some sensing devices with limited sensing ranges. This means at any time, a robot can only detect a local environment. In order to ensure that each robot can complete the given motion task with high efficiency and autonomy, we propose a fully distributed and real-time approach to trajectory planning. It is a method based on MPC strategy and mathematical programming. First, because of the local knowledge of the operating environment, we apply MPC strategy for each robot to update its detected environment and local valid trajectory in real time. Second, based on the detected environment on its current prediction horizon, a robot builds its own decoupled optimization subproblem, which may contain some parameters dependent on the future states of its neighboring robots. To construct its own problem independently, a robot makes a prediction of its neighbors' motion by communicating with its neighbors to retrieve their current and/or history states, rather than waiting for the prediction information from its neighbors. Third, the subproblem is solved via the iSCP method [98]. Since the building and solving of the subproblem do not rely on other robots' prediction, each robot can solve the subproblem and execute its motion in a distributed way. Fourth, once the time updates to the next horizon, the robot will update the environment and the communication with its new neighbors. Because of the great capability of description of mathematic programming, the proposed method is suitable for both 2D and 3D scenarios with any types of kinematics.

The main contribution of this work is a real-time and fully distributed trajectory planning method for multi-robot systems where each robot has no priori knowledge of the global environment. It has the following characteristics. 1) It uses MPC strategy to update the environment and update prediction results. Thus, each robot can plan its trajectory in real time. 2) It is fully distributed. Each robot communicates with its neighbors within its sensing range to retrieve some information which can be obtained immediately, such as the current states and the history records. With such information and the detected environment, robots can build and solve their own local subproblems independently. Thus, they can move in a distributed way without requiring the same parameter settings. 3) For each robot, the subproblem built at each time instant is resolved via the iSCP method, which is an improvement of SCP method. The significance of the proposed method is that each robot can both compute and move in a fully distributed way. This improves the flexibility and robustness of a multi-robot system, which are important in a multi-robot system. We also prove that the proposed method is with the minimal communication amount at each time instant.

This chapter is organized as follows. Section 4.2 states the problem addressed in this work. Sections 4.3 and 4.4 give the problem formalization and the algorithm to solve it, respectively. Section 4.5 gives some simulation results. Sections 4.6 and 4.7 give the discussion of the proposed method and the conclusion of our work, respectively.

4.2 Problem Statement

This part gives the problem statement of real-time trajectory planning in a multi-robot system. We first give a brief description of a multi-robot system, including the kinematics of robots, and then the problem statement. Suppose N is an integer indicating the number of robots, $\mathbb{I}_N = \{1, 2, \ldots, N\}$, and r_i, $i \in \mathbb{I}_N$, denotes robots. The motion space of each robot is in $\mathbb{R}^{n_0}$; $\boldsymbol{x}_i$, $\boldsymbol{v}_i$, and $\boldsymbol{a}_i$ are vectors in $\mathbb{R}^{n_0}$, denoting the position, velocity, and acceleration of robot r_i, respectively. If $\mathbb{R}^{n_0} = \mathbb{R}^2$, then it describes 2D scenarios such as for UGVs; while if $\mathbb{R}^{n_0} = \mathbb{R}^3$, then it is for 3D scenarios such as for UAVs. $[0, t_i]$ is the time interval for robot r_i to move.

Based on Definition 2 in Section 3.1, the state of robot r_i, denoted as $\boldsymbol{s}_i$, described in this chapter is a vector characterized by position, velocity, and acceleration, i.e., $\boldsymbol{s}_i = ({\boldsymbol{x}_i}^T, {\boldsymbol{v}_i}^T, {\boldsymbol{a}_i}^T)^T$, where $(\bullet)^T$ denotes the transposition operation of a vector. The set of all possible states of r_i forms the state space of r_i, denoted as $\mathcal{S}_i$. Suppose q_i is the trajectory of r_i in the time interval $[0, t_i]$. Then, $\forall \tau \in [0, t_i]$, the state of r_i at time τ is $q_i(\tau) = (\boldsymbol{x}_i(\tau)^T, \boldsymbol{v}_i(\tau)^T, \boldsymbol{a}_i(\tau)^T)^T \in \mathcal{S}_i$, where $\boldsymbol{x}_i(\tau)$, $\boldsymbol{v}_i(\tau)$, and $\boldsymbol{a}_i(\tau)$ are r_i's position, velocity, and acceleration at time τ, respectively.

Thus, to plan the trajectory of a robot is to determine the evolution of $\boldsymbol{x}_i$, $\boldsymbol{v}_i$, and $\boldsymbol{a}_i$. These measures are described by the kinematics of the robot. The kinematics of r_i is given by the following equations.

$$\dot{\boldsymbol{x}}_i(\tau) = \boldsymbol{v}_i(\tau), \dot{\boldsymbol{v}}_i(\tau) = \boldsymbol{a}_i(\tau), \forall \tau \in [0, t_i]; \tag{4.1}$$

$$\boldsymbol{x}_i(0) = \boldsymbol{x}_0^i, \boldsymbol{x}_i(t_i) = \boldsymbol{x}_f^i, \boldsymbol{v}_i(0) = \boldsymbol{0}, \boldsymbol{v}_i(t_i) = \boldsymbol{0}; \tag{4.2}$$

$$\boldsymbol{v}_i(\tau) \in [\boldsymbol{v}_{\min}, \boldsymbol{v}_{\max}], \boldsymbol{a}_i(\tau) \in [\boldsymbol{a}_{\min}, \boldsymbol{a}_{\max}]. \tag{4.3}$$

where $\boldsymbol{x}_0^i$ and $\boldsymbol{x}_f^i$ are the initial and target positions of r_i, respectively; $\dot{\boldsymbol{x}}_i(\tau)$ and $\dot{\boldsymbol{v}}_i(\tau)$ are the derivatives of $\boldsymbol{x}_i(\tau)$ and $\boldsymbol{v}_i(\tau)$ with respect to the time variable τ, respectively; $\boldsymbol{v}_{\min}$ and $\boldsymbol{v}_{\max}$ are the vectors containing the lower and upper bounds of each velocity component, respectively. So are $\boldsymbol{a}_{\min}$ and $\boldsymbol{a}_{\max}$ for each acceleration component. Note that for such kinematics, the control inputs are its acceleration.

TABLE 4.1: Summarization of Symbols in This Chapter

Symbols	Meanings
$\boldsymbol{x}_0^i$, $\boldsymbol{x}_f^i$	The initial and target positions of robot r_i, respectively.
t_i, h_i, T_i	The required arrival time, discrete time step, and the number of discrete instants of r_i, respectively; $t_i = T_i h_i$.
$\boldsymbol{x}_i[k]$, $\boldsymbol{v}_i[k]$, $\boldsymbol{a}_i[k]$	The position, velocity, and acceleration of r_i at the discrete time instant k, $k \in \{1, 2, \ldots, T_i\}$.
$\boldsymbol{a}_i[k_0 : T_i - 1]$	The sequence of accelerations from k_0 to $T_i - 1$: $\boldsymbol{a}_i[k_0]$, $\boldsymbol{a}_i[k_0 + 1]$, . . ., $\boldsymbol{a}_i[T_i - 1]$.
H_i	The length of prediction horizon of r_i.
k_0	The current time instant.
k_{0,H_i}	The end time instant of the current prediction horizon, $k_{0,H_i} = \min\{k_0 + H_i, T_i\}$.
$\mathcal{H}_i[k_0]$	The set of discrete time instants in the current prediction horizon, $\mathcal{H}_i[k_0] = \{k_0 + 1, \ldots, k_{0,H_i}\}$.
$\mathcal{O}_i^\alpha[k_0]$, $\mathcal{O}_i^\beta[k_0]$	The sets of static and dynamic obstacles detected by r_i at the current instant k_0.
ρ	The safe radius of robots.

Suppose a multi-robot system contains N robots, whose kinematics are described by (4.1)–(4.3). Different robots are placed at different initial positions $\boldsymbol{x}_0^i$, and they need to move to different target positions $\boldsymbol{x}_f^i$. Each robot has no priori knowledge of the operating environment, and the environment may be changing. Moreover, the robots are equipped with sensors to detect the environment. Suppose the sensing range is L, and thus each robot can only detect the environment within its sensing range. Recall that in our basic assumptions described in Section 3.1, each robot can detect and locate other robots and obstacles within its sensing range.

In the sequel, we give the problem resolved in this chapter:

Problem 1. Given a multi-robot system with N robots, whose kinematic equations are given in (4.1)–(4.3), initial and target positions are $\boldsymbol{p}_0 = \{\boldsymbol{x}_0^1, \boldsymbol{x}_0^2, \ldots, \boldsymbol{x}_0^N\}$ and $\boldsymbol{p}_f = \{\boldsymbol{x}_f^1, \boldsymbol{x}_f^2, \ldots, \boldsymbol{x}_f^N\}$, satisfying $\forall i, j \in \mathbb{I}_N$, $i \neq j$, $\boldsymbol{x}_0^i \neq \boldsymbol{x}_0^j$ and $\boldsymbol{x}_f^i \neq \boldsymbol{x}_f^j$, find a trajectory for each robot r_i such that it can move from $\boldsymbol{x}_0^i$ to $\boldsymbol{x}_f^i$ without causing any collisions with other robots and obstacles in the environment.

Before giving the solution of this problem, we first show in Table 4.1 the symbols used in this chapter.

4.3 Formal Modeling for the Problem

In this section, we model our problem as a distributed optimization programming.

4.3.1 Problem Analysis

First of all, we analyze the main challenges of the problem described in Section 4.2 to have a better understanding.

First, local environmental knowledge and local available kinematics. Due to the limitation of the sensing and communication ranges, each robot can only detect some limited information of the motion environment. Let us take a wheeled robot in 2D as an example. But this does not mean that it is only for wheeled robots. Indeed, as described before, it can be for both 2D and 3D scenarios. As shown in Fig. 4.1, the sets of obstacles detected by the robot at x_1, x_2, and x_3 are $\{o_1, o_2, o_3\}$, $\{o_3, o_4\}$, and $\{o_4, o_5, o_6\}$, respectively. Moreover, if the environment can change dynamically, a robot may also find different obstacles within the same sensing range. Thus, with the current information, only the trajectory in the current sensing region is available at the current time instant. This means that at each time instant, only the kinematics within the current sensing range is available. Note that such limited information may not lead a robot to its target, which is regarded as the problem of completeness or convergency. In this chapter, MPC strategy is applied to detect obstacles in real time. To guarantee the convergency, the whole kinematics is taken into consideration even though the kinematics beyond the sensing range is unreliable.

Second, fully distributed motion. In a multi-robot system, multiple robots are moving in a shared environment simultaneously, and different robots may execute different tasks. In order to improve flexibility of the system, robots are expected to move in a fully distributed way. This implies that each robot needs to plan its trajectory and execute its motion only by communicating with its neighbors to retrieve some directly obtained information, rather than by waiting for others' prediction information since 1) in some cases, to obtain such information, a robot has to wait for other robots to finish

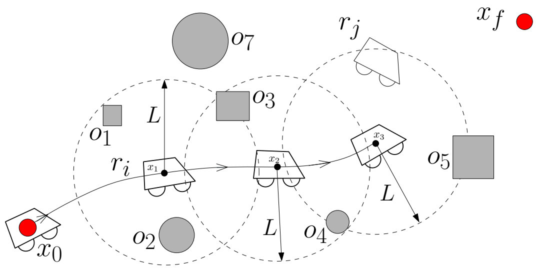


FIG. 4.1: A robot moves with a limited sensing range. x_0 and x_f are the initial and target positions, respectively. The dashed circles are the boundaries of sensing area, where L is the sensing range. $o_1 - o_7$ are obstacles in the environment. The robot detects different obstacles at different positions. The robot detects obstacles $o_1 - o_3$ when it is at x_1, o_3 and o_4 at x_2, and $o_4 - o_6$ at x_3.

their computation; and 2) because of the different time discretization steps, the predicted information by other robots may not be the required one at the time instants of this robot. Thus, a robot needs to predict the future motion of its neighbors to avoid collision during the current prediction horizon. In this work, a robot adopts a linear method to predict its neighbors' positions by assuming they are doing uniform linear motion in its current prediction horizon. We say it is reasonable for the following two reasons: 1) Because of the physical laws, the velocity cannot change suddenly; 2) Only the positions at the next time instant are functional since a robot needs to re-predict others' motion once it moves to the next time instant.

Third, optimization criteria. Even in a clustered environment, a robot may have lots of feasible trajectories to its target. Because of some requirements or limitations, there are some criteria required, such as minimum distance, minimum rotation, and so on [15]. For example, if the motion task is urgent, then a robot is expected to move with minimum time; if the energy is limited, then a robot should move with minimum control effort. Hence, a planning algorithm should be able to adjust to different optimization criteria. We deal with motion planning via mathematical optimization since we can formulate different optimization objectives to much extent.

Considering the above requirements, we in the sequel give the detailed model for the real-time motion planning.

4.3.2 Construction of Distributed Optimization Programming

In this subsection, we give the optimization formulation of the trajectory planning problem. First, we give the discrete-time forms of (4.1)−(4.3).

Suppose the time interval of robot r_i, $i \in \mathbb{I}_N$, is discretized into T_i discrete time instants with an equal time step h_i, i.e., $0 = \tau_0, \tau_1, \ldots, \tau_{T_i} = t_i$, where $\tau_k = kh_i$ for $k = 0, 1, 2, \ldots, T_i$. Note that the selection of h_i should satisfy $\|\boldsymbol{v}_{\max} h_i\|_2 \le L$, where $\|\cdot\|_2$ is the 2-norm in the Euclidean space and L is the sensing range. This condition guarantees that the first predicted position is within the sensing range and thus available. The discrete-time kinematics of r_i at the current time instant k_0 can be described as:

$$\boldsymbol{x}_i[k+1] = \boldsymbol{x}_i[k] + \boldsymbol{v}_i[k]h_i + \frac{\boldsymbol{a}_i[k]}{2}h_i^2, \tag{4.4}$$

$$\boldsymbol{v}_i[k+1] = \boldsymbol{v}_i[k] + \boldsymbol{a}_i[k]h_i, \tag{4.5}$$

$$\boldsymbol{v}_i[k] \in [\boldsymbol{v}_{\min}, \boldsymbol{v}_{\max}], \boldsymbol{a}_i[k] \in [\boldsymbol{a}_{\min}, \boldsymbol{a}_{\max}]. \tag{4.6}$$

$$\boldsymbol{x}_i[T_i] = \boldsymbol{x}_f^i, \boldsymbol{v}_i[T_i] = \boldsymbol{0}, \tag{4.7}$$

$$k = k_0, \ldots, T_i - 1.$$

where $\boldsymbol{x}_i[k_0]$ and $\boldsymbol{v}_i[k_0]$ are the current position and velocity of r_i, respectively.

In the sequel, we give the procedure to construct the final optimization problem of the multi-robot system at the current time instant k_0. Note that H_i is the length of the prediction horizon of r_i; the discrete time instants in the current horizon are denoted as $\mathcal{H}_i[k_0] = \{k_0 + 1, \ldots, k_{0,H_i}\}$, where $k_{0,H_i} = \min\{k_0 + H_i, T_i\}$. For simplicity, we use $\boldsymbol{x}_i(\mathcal{H}_i[k_0])$ to denote r_i's positions at the time instants in $\mathcal{H}_i[k_0]$, i.e., $\boldsymbol{x}_i(\mathcal{H}_i[k_0]) = \{\boldsymbol{x}_i[k_0 + 1], \ldots, \boldsymbol{x}_i[k_{0,H_i}]\}$.

Decoupled Objective Function. A typical objective function for a multi-robot system with N robots at the current time instant k_0 can be described as follows.

$$f = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k1=k_0}^{T_i-1}\sum_{k2=k_0}^{T_j-1} w_{ij}\boldsymbol{a}_i^T[k1]\boldsymbol{a}_j[k2] \tag{4.8}$$

where w_{ij}, $i, j \in \mathbb{I}_N$, are scalar weights. By choosing different weights, this objective function can describe different measures, such as the total path length, i.e., $\sum_{i=1}^{N} \sum_{k=k_0}^{T_i-1} \|\boldsymbol{x}_i[k+1] - \boldsymbol{x}_i[k]\|_2$, and the control effort of the system, i.e., $\sum_{i=1}^{N} \sum_{k=k_0}^{T_i-1} \|\boldsymbol{a}_i[k]\|_2$.

In the general case, the function in (4.8) couples the accelerations of all robots. This makes the problem centralized rather than distributed. Hence, we need to decouple (4.8) so that each robot can optimally predict its trajectory autonomously. To make sure that r_i only needs to determine its own optimal accelerations, i.e., $\boldsymbol{a}_i[k_0 : T_i - 1]$, we set $w_{ij} = 1$ for $i = j$ and $w_{ij} = 0$ for $i \neq j$ in (4.8). So the decoupled objective function for r_i is given in (4.9).

$$f_{obj}^i = \sum_{k=k_0}^{k_{0,H_i}-1} \|\boldsymbol{a}_i[k]\|_2^2 + \sum_{k=k_{0,H_i}}^{T_i-1} \|\boldsymbol{a}_i[k]\|_2^2 \tag{4.9}$$

where $k_{0,H_i} = \min\{k_0 + H_i, T_i\}$. Hence, the optimal objective for r_i is:

$$\min_{\boldsymbol{a}_i[k_0:T_i-1]} f_{obj}^i = \sum_{k=k_0}^{k_{0,H_i}-1} \|\boldsymbol{a}_i[k]\|_2^2 + \sum_{k=k_{0,H_i}}^{T_i-1} \|\boldsymbol{a}_i[k]\|_2^2.$$

We say this individual objective is meaningful since it means that robot r_i wants to induce a smooth trajectory with the lowest curvatures. This objective function contains two parts for $k_0 + H_i < T_i$. The first one is with respect to the acceleration on the prediction horizon, which ensures collision avoidance with the observed obstacles; while the second one is a penalty term that can guarantee the global target convergence, i.e., the robot can arrive at its target at the end of the given time interval. From this optimal objective, we find that the only optimal variable for each robot is its acceleration.

Kinematic Constraints with Target Convergence. The discrete-time kinematic constraints at time instant k_0 are described by (4.4)–(4.7). The kinematics can also be divided into two kinds when $k_0 + H_i < T_i$. The first one is the desired kinematics on the prediction horizon, i.e., $\forall k \in \mathcal{H}_i[k_0]$; while the second one is the kinematics beyond the horizon, i.e., $k = k_0 + H_i, \ldots, T_i - 1$. The former one is the available kinematics since they are considered to avoid collisions with the observed environmental obstacles and other robots. The latter one is for the global target convergence, and they do not need to consider any collision avoidance constraints.

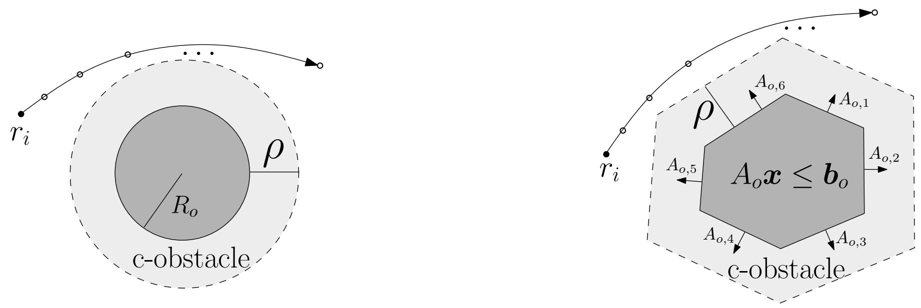
Collision Avoidance Constraints with Local Detection. Now we give the constraints for collision avoidance. Because of the change of the environment, a robot may observe different obstacles, including the external obstacles and other robots, in its sensing range at different time instants. Suppose $\mathcal{O}_i[k_0]$ is the set of obstacles that r_i detects at time instant k_0. It can be divided into two kinds. The first one is the set of static obstacles, denoted as $\mathcal{O}_i^{\alpha}[k_0]$. A static obstacle is an object with zero velocity forever. At any time, r_i can locate the positions of the static obstacles in its sensing range. The second one is the set of dynamic obstacles, including other robots, denoted as $\mathcal{O}_i^{\beta}[k_0]$. A dynamic obstacle is an object with a non-zero velocity. At any time, r_i can retrieve the positions and velocities of the dynamic obstacles in its sensing range. With this information, r_i further needs to predict their positions in the future. Clearly, we have $\mathcal{O}_i[k_0] = \mathcal{O}_i^{\alpha}[k_0] \cup \mathcal{O}_i^{\beta}[k_0]$ and $\mathcal{O}_i^{\alpha}[k_0] \cap \mathcal{O}_i^{\beta}[k_0] = \emptyset$.

Suppose each robot has a safe radius ρ, and thus it is modeled as a sphere. By enlarging the obstacles with the safe radius, we can construct c-obstacles, as well as the configuration space. In this way, each robot can be regarded as a point. Usually, Minkowski sum can be used to construct the c-obstacles [13]. However, in order to preserve the polyhedral shapes, we apply another way to build the c-obstacles of polyhedral obstacles, which is described later.

Remark 1. In practice, a robot may not locate at the given positions exactly because of some uncertain events, such as the sliding of wheels. However, such derivation should not be large. Suppose the maximal tolerable derivation is δ. We can address this derivation by adding δ to the original safe radius ρ_0. So the actual safe radius is $\rho = \rho_0 + \delta$. If the deviation is larger than δ, we need to check the physical conditions of the robot manually.

Next, we give the details of the collision avoidance constraints. Recall that $k_{0,H_i} = \min\{k_0 + H_i, T_i\}$ and $\mathcal{H}_i[k_0] = \{k_0 + 1, \ldots, k_{0,H_i}\}$.

Collision avoidance with static obstacles. Usually, most static obstacles in the environment are with irregular shapes such that we cannot describe them in closed-form



(a) Collision avoidance with a static spherical obstacle

(b) Collision avoidance with a static polyhedral obstacle

FIG. 4.2: Collision avoidance with different shapes of static obstacles. The gray regions represent the obstacles and the dashed boundaries are the safe boundaries.

expressions. Thus, we first need to model them with approximate regular shapes. In this work, a static obstacle is approximated as a sphere or polyhedron. Suppose $\mathcal{O}_i^{\alpha_1}[k_0]$ and $\mathcal{O}_i^{\alpha_2}[k_0]$ are the sets of spherical and polyhedral approximations of the static obstacles, respectively. Thus, $\mathcal{O}_i^{\alpha}[k_0] = \mathcal{O}_i^{\alpha_1}[k_0] \cup \mathcal{O}_i^{\alpha_2}[k_0]$ and $\mathcal{O}_i^{\alpha_1}[k_0] \cap \mathcal{O}_i^{\alpha_2}[k_0] = \emptyset$. Fig. 4.2 shows collision avoidance with different shapes of static obstacles.

For a spherical obstacle o, the collision avoidance constraints can be written as $\|\boldsymbol{x}_i[k] - \boldsymbol{x}_o\|_2 \geq \rho_o + \rho$, where $\boldsymbol{x}_o$ and ρ_o are the center and radius of obstacle o, respectively. Thus, the collision avoidance constraints with respect to the spherical obstacles can be given by (4.10).

$$\begin{aligned} &\forall o \in \mathcal{O}_i^{\alpha_1}[k_0], \forall k \in \mathcal{H}_i[k_0], \\ &\|\boldsymbol{x}_i[k] - \boldsymbol{x}_o\|_2 \geq \rho_o + \rho. \end{aligned} \tag{4.10}$$

A polyhedral obstacle o is modeled by a set of linear inequalities, i.e., $\{\boldsymbol{x} | A_o\boldsymbol{x} \leq \boldsymbol{b}_o\}$, where $A_o = (A_{o,1}^T, \ldots, A_{o,m}^T)^T$ and $\boldsymbol{b}_o = (b_{o,1}, \ldots, b_{o,m})^T$ are a matrix and a vector with proper dimensions, respectively. So the collision avoidance constraints for such obstacles are:

$$\begin{aligned} &\forall o \in \mathcal{O}_i^{\alpha_2}[k_0], \forall k \in \mathcal{H}_i[k_0], \\ &\boldsymbol{x}_i[k] \notin \{\boldsymbol{x} | A_o\boldsymbol{x} < \boldsymbol{b}_o + \|A_o\|_{\bullet}\rho\}. \end{aligned} \tag{4.11}$$

where $\|A_o\|_{\bullet} = (\|A_{o,1}\|_2, \ldots, \|A_{o,m}\|_2)^T$.

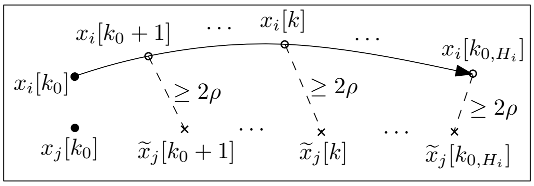


FIG. 4.3: Illustration of collision avoidance with r_j from time k_0. The points with crosses are r_j's positions predicted by r_i. The distance between the two positions with the same time should not be less than 2ρ.

Collision avoidance with dynamic obstacles. Here we only consider a simple situation: The only dynamic obstacles are the robots. At each time, the robots that are needed to be avoided by robot r_i are the robots within r_i's sensing range, i.e., $\mathcal{O}_i^\beta[k_0] = \{r_j | \|\boldsymbol{x}_j[k_0] - \boldsymbol{x}_i[k_0]\|_2 \leq L\}$. To avoid collisions with other robots, r_i first needs to retrieve the current states of the robots in $\mathcal{O}_i^\beta[k_0]$ and predict their future positions. Some work deals with the prediction by increasing the shapes of robots according to their maximal speeds and initial positions [198]. However, this is too conservative to forbid some feasible space. Here, we introduce a linear prediction method. Suppose the current position and velocity of r_j, $r_j \in \mathcal{O}_i^\beta[k_0]$, are $\boldsymbol{x}_j[k_0]$ and $\boldsymbol{v}_j[k_0]$, respectively. Thus, r_i can predict the motion of r_j to be uniform motion. In this way, r_j's future positions predicted by r_i are computed as follows.

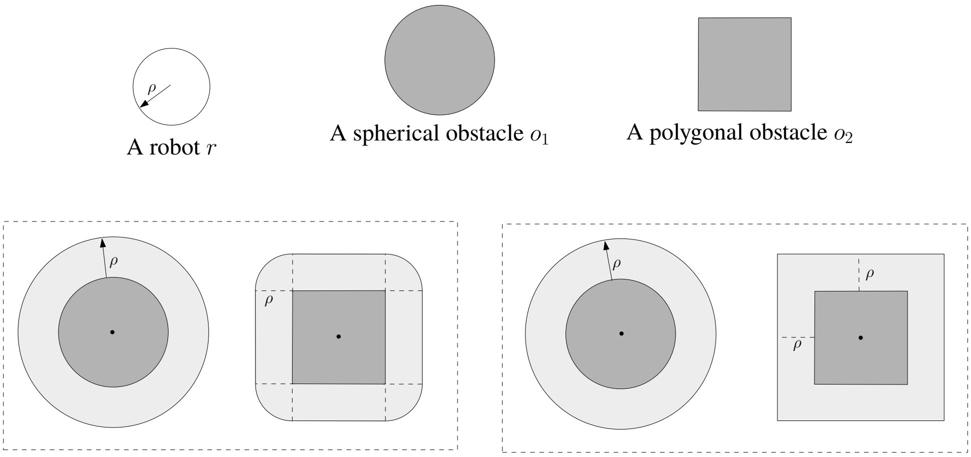
$$\widetilde{\boldsymbol{x}}_j[k] = \boldsymbol{x}_j[k_0] + (k - k_0)\boldsymbol{v}_j[k_0]h_i, \quad \forall k \in \mathcal{H}_i[k_0] \tag{4.12}$$

where h_i is the discrete time step of r_i. In this way, we can find that there is no need to synchronize time discretization among robots.

Fig. 4.3 illustrates this idea for r_i to avoid collisions with r_j from the current time k_0. Hence, r_i's constraints for collision avoidance with the robots in $\mathcal{O}_i^\beta[k_0]$ at k_0 can be described in (4.13).

$$\begin{aligned} &\forall r_j \in \mathcal{O}_i^\beta[k_0], \forall k \in \mathcal{H}_i[k_0], \\ &\|\boldsymbol{x}_i[k] - \widetilde{\boldsymbol{x}}_j[k]\|_2 \geq 2\rho. \end{aligned} \tag{4.13}$$

With its own estimation of the neighboring robots, a robot can avoid the situation that two robots move directly to each other, which will cause a deadlock. In fact, because of the uniform motion prediction of its neighbors, a robot regards the region



(a) The Makowski sum of o_1 and r, and o_2 and r.

(b) The c-obstacles of o_1 and o_2 built in our work.

FIG. 4.4: Comparison of c-obstacles built by our work and by Minkowski sum in 2D space.

ahead of it as a collision region when it detects that another robot is moving directly to it. Then it plans its own trajectory to deviate from its original direction. In this way, the two robots will be separated before they are too close to change their directions. For robots that cannot change their move directions, DES models can be applied to detect and avoid deadlocks, which will be studied in the following chapters. Thus, collisions and deadlocks can always be avoided. Besides, when all the configuration parameters of the robots in a system are the same, by negotiating with its neighbors, a robot can avoid the situation that two or more robots are computing collision-free trajectories simultaneously. Thus, livelocks, namely some robots keep moving but will never reach their targets, can be avoided. Demonstrative examples will be illustrated in Section 4.5.3.

At last, we give the comparison to build the c-obstacles with Minkowski sum [14] and with our method, which is shown in Fig. 4.4. In Fig. 4.4, the first row shows the original models of a robot, a spherical obstacle, and a polygonal obstacle in the 2D space, respectively; Fig. 4.4(a) gives the Minkowski sum of o_1 and r, and o_2 and r, respectively; and Fig. 4.4(b) shows the c-obstacles constructed in our work. Clearly, our method can keep the regular shapes of obstacles.

In conclusion, for any robot r_i, $i \in \mathbb{I}_N$, its local optimization problem at instant k_0, denoted as $P_i[k_0]$, can be described as follows.

$$(P_i[k_0]) \qquad \min_{\boldsymbol{a}_i[k_0:T_i-1]} f^i_{obj}$$
$$\text{subject to (4.4)–(4.7), (4.10), (4.11), and (4.13).}$$

Remark 2. In each sub-problem $P_i[k_0]$, the collision avoidance constraints can be divided into $k_{0,H_i} - k_0$ kinds. The q-th kind of constraints are the constraints that position $\boldsymbol{p}_i[k_0 + q]$ should satisfy. It mandates the following constraints: $\|\boldsymbol{x}_i[k_0 + q] - \boldsymbol{x}_o\|_2 \geq \rho_o + \rho$ for all $o \in \mathcal{O}_i^{\alpha_1}[k_0]$, $\boldsymbol{x}_i[k_0 + q] \notin \{\boldsymbol{x}|A_o\boldsymbol{x} < \boldsymbol{b}_o + \|A_o\|_\bullet\rho\}$ for all $o \in \mathcal{O}_i^{\alpha_2}[k_0]$, and $\|\boldsymbol{x}_i[k_0 + q] - \widetilde{\boldsymbol{x}}_j[k_0 + q]\|_2 \geq \rho$ for all $r_j \in \mathcal{O}_i^{\beta}[k_0]$. We say this kind of constraints is with respect to $\boldsymbol{x}_i[k]$.

4.3.3 Distributivity Analysis

In this subsection, we describe the distributed nature of the built optimization problem. During the construction of $P_i[k_0]$, r_i needs to search the environment within its sensing range and communicate with the robots detected. For example, Fig. 4.5 gives a motion planning framework of a multi-robot system with three robots. First, each robot has its sensors to monitor the environment. In this scenario, r_1 at $\boldsymbol{x}_1$ detects r_2 is in its sensing range, while r_2 at $\boldsymbol{x}_2$ detects r_1 and r_3, and r_3 at $\boldsymbol{x}_3$ detects r_2. Second, a robot communicates with the robots within its sight to build the local optimization problem. r_1 only needs to communicate with r_2, and r_3 also only needs to communicate with r_2, while r_2 needs to communicate with r_1 and r_3. The communication among them is described by the dashed arrows in Fig. 4.5. Third, the robot solves the optimization subproblem and actuates its motion. It should also be able to broadcast its position and velocity to its neighbors.

In Fig. 4.5, take r_2 as an example to explain how a robot can plan its trajectory only with some communication with other robots. First, r_2 searches for obstacles in the environment within its sensing, i.e., the region bounded by the middle circle. Since it finds r_1 and r_3, r_2 sends requests to communicate with them and then receives their current

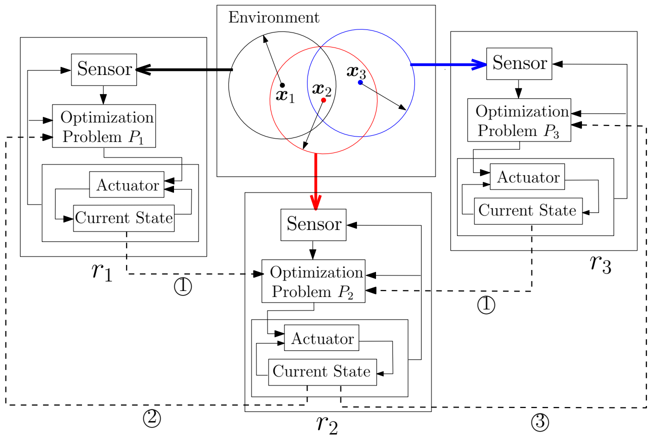


FIG. 4.5: The framework of distributed trajectory planning for a multi-robot system. $\boldsymbol{x}_1 - \boldsymbol{x}_3$ are the current positions of three robots, and the circles are the local environments detected by the corresponding robots.

positions and velocities. The process is represented by the dashed arrows marked by "①". Second, using the received information, r_2 constructs and solves its optimization subproblem P_2 individually. Third, the first predicted control signal is sent to the actuator, and r_i executes the corresponding move. During the move, if it receives the requests for communication, r_2 sends its current position and velocity to the senders, such as the communication marked by "②" and "③" in Fig. 4.5.

Theorem 1. The proposed trajectory planning approach is of the minimal amount of communication.

Proof. According to the framework, a robot first needs to detect the environment. This process can be done independently for each robot. So there is no communication among robots. The second process is to communicate with the robots within its sensing range. Such communication is to retrieve the current positions and velocities of other robots. Note that for any algorithms, if we want to avoid collision with one robot, we need at least the knowledge of its current position. Thus, such communication cannot be avoided in order to be safe for any non-centralized approaches. After obtaining such information, the robot can independently build its optimization problem, solve it, and finally actuate its motion. There is no other communication for these operations. In conclusion, the proposed trajectory planning method is of the minimum amount of communication. □

Indeed, the communication complexity is $O(N)$, where N is the number of robots in the system. However, each time a robot only needs to communicate with robots within its communication range.

4.4 Real-Time Trajectory Planning Algorithm

In the above section, we construct the distributed optimization programming for the system at an arbitrary time instant. In this section, we describe the algorithm to solve Problem 1.

The general principle of our MPC based real-time trajectory planning algorithm is that: For an arbitrary robot r_i, at the current time instant k_0, it predicts its future control signals, i.e., accelerations, on the current prediction horizon by solving its subproblem $P_i[k_0]$; then the first control signal is applied while others are discarded; when the system reaches the next state, the horizon recedes to the next one and the future control signals are regenerated from the updated optimization problem.

Usually, $P_i[k_0]$ is non-convex and thus it is hard to find a global optimal solution. Fortunately, the theory of convex programming gives the inspiration to solve nonlinear optimization problems approximately and efficiently. In this work, we solve the optimization problems $P_i[k_0]$ via iSCP, which is an extension of the SCP method. This is because compared with the SCP method, iSCP has higher probability to find a feasible trajectory during the approximate solving of the optimization problem [98]. The procedure of iSCP contains the following steps.

Step 1 : Initialization. Set the initial values of $\boldsymbol{x}_i[k_0{+}1 : T_i]$, denoted as ${}^0\boldsymbol{x}_i[k_0{+}1 : T_i]$; set the collision avoidance constraints to be empty; $m = 0$.

Step 2 : Search for the first position whose value violates the collision constraints; and add the constraints with respect to this position into the optimization problem. Note that these kinds of constraints will be always included in the future. Thus, this position always satisfies the collision constraints at the following iterations.

Step 3 : Convexify the constraints to obtain an approximate convex problem and then solve it. The solution is denoted as ${}^{m+1}\boldsymbol{x}_i[k_0+1:T_i]$.

Step 4 : If ${}^{m+1}\boldsymbol{x}_i[k_0+1:T_i]$ satisfies all collision constraints and $\forall k \in \{k_0+1,\ldots,T_i\}$, $\|{}^{m+1}\boldsymbol{x}_i[k] - {}^{m}\boldsymbol{x}_i[k]\|_\infty \leq \epsilon$, stop and return ${}^{m+1}\boldsymbol{x}_i[k_0+1]$, ${}^{m+1}\boldsymbol{v}_i[k_0+1]$, and ${}^{m+1}\boldsymbol{a}_i[k_0]$. Otherwise, $m = m+1$ and go back to Step 2.

4.4.1 Convexification of the Non-Convex Constraints

The main challenge of iSCP is to approximate the nonlinear problem $P_i[k_0]$ by a convex problem at Step 3. In $P_i[k_0]$, the non-convex constraints are the inequalities of (4.10), (4.11), and (4.13). So we need to convexify them. There are two forms of these inequality constraints, i.e., those given in the form of (4.10) or (4.13), and those given in the form of (4.11).

At iteration $m+1$ of iSCP, using the first order Taylor approximation, the first form can be approximated by the following formula.

$$\frac{({}^{m}\boldsymbol{x}_i[k] - \overline{\boldsymbol{x}})^T}{\|{}^{m}\boldsymbol{x}_i[k] - \overline{\boldsymbol{x}}\|_2}(\boldsymbol{x}_i[k] - \overline{\boldsymbol{x}}) \geq \rho' \tag{4.14}$$

where $k \in \mathcal{H}_i[k_0]$, $\rho' = \rho + \rho_0$ and $\overline{\boldsymbol{x}} = \boldsymbol{x}_o$ for (4.10), $\rho' = 2\rho$ and $\overline{\boldsymbol{x}} = \widetilde{\boldsymbol{x}}_j[k]$ for (4.13); $\boldsymbol{x}_i[k]$ is a linear combination of the optimal variables $\boldsymbol{a}_i[k_0], \boldsymbol{a}_i[k_0+1], \ldots, \boldsymbol{a}_i[k-1]$; ${}^{m}\boldsymbol{x}_i[k]$ and $\widetilde{\boldsymbol{x}}_j[k]$, whose values are known at $m+1$, are the solution to $\boldsymbol{x}_i[k]$ at the m-th iteration and r_i's prediction for the positions of r_j, respectively. Indeed, since function $\|\boldsymbol{x} - \overline{\boldsymbol{x}}\|_2$ is a convex function, for any $\boldsymbol{x}$ and ${}^{m}\boldsymbol{x}$, we have $\|\boldsymbol{x} - \overline{\boldsymbol{x}}\|_2 \geq \|{}^{m}\boldsymbol{x} - \overline{\boldsymbol{x}}\|_2 + \frac{({}^{m}\boldsymbol{x}-\overline{\boldsymbol{x}})^T}{\|{}^{m}\boldsymbol{x}-\overline{\boldsymbol{x}}\|_2}(\boldsymbol{x} - {}^{m}\boldsymbol{x}) \geq \frac{({}^{m}\boldsymbol{x}-\overline{\boldsymbol{x}})^T({}^{m}\boldsymbol{x}-\overline{\boldsymbol{x}}+\boldsymbol{x}-{}^{m}\boldsymbol{x})}{\|{}^{m}\boldsymbol{x}-\overline{\boldsymbol{x}}\|_2} = \frac{({}^{m}\boldsymbol{x}-\overline{\boldsymbol{x}})^T}{\|{}^{m}\boldsymbol{x}-\overline{\boldsymbol{x}}\|_2}(\boldsymbol{x} - \overline{\boldsymbol{x}})$. This means once (4.14) is satisfied, (4.10) and (4.13) are always satisfied. Hence, the trust region for such an approximation is the whole space. Fig. 4.6 gives an example of a convex feasible region of $\boldsymbol{x}_i[k]$, i.e., the quadrilateral region, based on the above convex approximation. In this case, robot r_i at time k should avoid collisions with three robots and a spherical obstacle.

Next, let's consider the second one, i.e., the inequalities in the form of (4.11). Suppose a polyhedral obstacle is represented as $o = \{\boldsymbol{x} | A_o\boldsymbol{x} \leq \boldsymbol{b}_o\}$, where $A_o = (A_{o,1}^T, \ldots,$

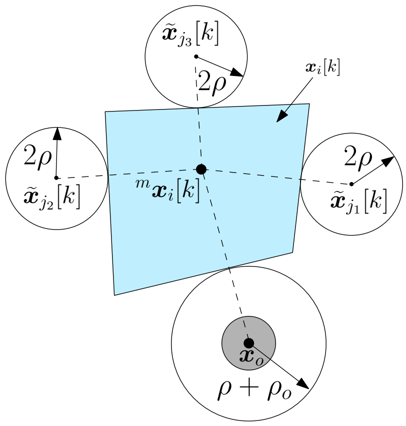


FIG. 4.6: Robot r_i's approximate collision-free region for the future position $\boldsymbol{x}_i[k]$ at iteration m. The robot at this position should avoid three robots $r_{j_1} - r_{j_3}$ and a static spherical obstacle o. The quadrilateral region is the convexified configuration space of $\boldsymbol{x}_i[k]$. Each boundary of this region is tangent to the corresponding circle.

$A_{o,z}^T)^T$ and $\boldsymbol{b}_o = (b_{o,1}, \ldots, b_{o,z})^T$. For such an obstacle, we can select one boundary as the reference hyperplane, and guarantee that the feasible positions are on the other side of the hyperplane with the safe distance ρ. Thus, the approximate convex set for (4.11) can be described as follows.

$$A_{o,z_0}^T \boldsymbol{x}_i[k] \geq b_{o,z_0} + \|A_{o,z_0}\|_2 \rho \tag{4.15}$$

where z_0, $z_0 \in \{1, \ldots, z\}$, is the index of the selected reference hyperplane.

There are many possible ways that can be applied to select the reference hyperplane of the obstacle $o = \{\boldsymbol{x} | A_o \boldsymbol{x} \leq \boldsymbol{b}_o\}$. We give a heuristic approach considering the physical limitations of robots and the avoidance of over-moving. The boundaries of the polyhedral obstacle o, denoted as $o_1, \ldots, o_z$, are described as $o_{z'} = \{\boldsymbol{x} | A_{o,z'}^T \boldsymbol{x} = b_{o,z'}\}$, $z' = 1, 2, \ldots, z$. The corresponding safe boundary of $o_{z'}$ can be described as $\{\boldsymbol{x} | A_{o,z'}^T \boldsymbol{x} = b_{o,z'} + \|A_{o,z'}\|_2 \rho\}$. The selection procedure can be described as follows. First, determine the hyperplane in front of it, say $o_{z_1} = \{\boldsymbol{x} | A_{o,z_1}^T \boldsymbol{x} = b_{o,z_1}\}$. Clearly, $\boldsymbol{x}_i[k_0]$ satisfies $A_{o,z_1}^T \boldsymbol{x}_i[k_0] \geq b_{o,z_1} + \|A_{o,z_1}\|_2 \rho$. Second, search for the hyperplanes that are adjacent to o_{z_1}. Two hyperplanes are said to be adjacent if their intersection is not empty. Third, among these adjacent hyperplanes, select one hyperplane, say $o_{z_0} = \{\boldsymbol{x} | A_{o,z_0}^T \boldsymbol{x} = b_{o,z_0}\}$, such that (i) the obstacle o and r_i's current position are at the same side of o_{z_0}, and (ii) it is the closest boundary to the target position. Thus, o_{z_0} becomes the reference hyperplane. Note that the adjacency of the selected hyperplane

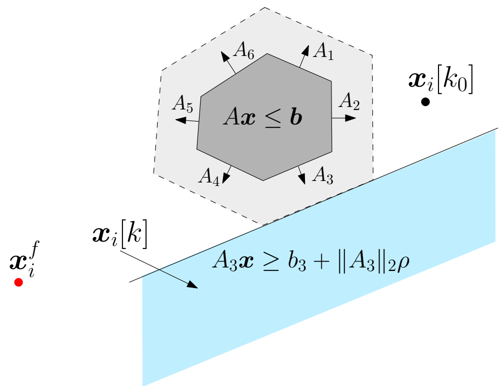


FIG. 4.7: The convexification of the polyhedral collision constraints. $A_1 - A_6$ are the outward normal vectors of polyhedron o. The boundary with the outward normal vector A_3 is selected as the reference hyperplane based on our heuristic method. The bottom-right region is the convex approximation of the collision avoidance for the future position $\boldsymbol{x}_i[k]$.

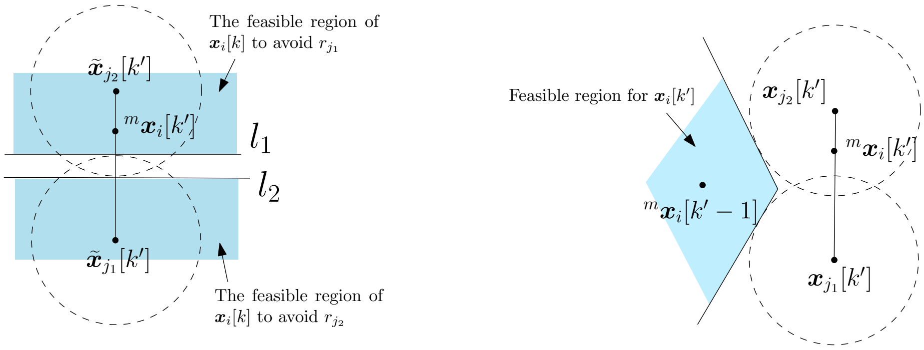
guarantees that the future position $\boldsymbol{x}_i[k]$ cannot be too far away from $\boldsymbol{x}_i[k_0]$; while the different sides of $\boldsymbol{x}_i[k]$ and $\boldsymbol{x}_i[k_0]$ can help to avoid over-moving.

For example, Fig. 4.7 shows an example of the convex approximation of a polyhedral obstacle in the 2D space. The polyhedron is described as $o = \{\boldsymbol{x}|A_z^T\boldsymbol{x} \leq b_z, z = 1, \ldots, 6\}$, where $A_1 - A_6$ are the outward normal vectors of the boundaries of the polyhedron, and the corresponding boundaries are denoted as $o_1 - o_6$. At the current time instant, the boundary in front of the robot is o_2: $A_2^T\boldsymbol{x} = b_2 + \|A_2\|_2\rho$. Clearly, $A_2^T\boldsymbol{x}_i[k_0] \geq b_2 + \|A_2\|_2\rho$. Its adjacent hyperplanes are o_1 and o_3. For either one, the obstacle and the current position are at the same side of the corresponding boundary. However, o_3 is selected as the reference plane since the target position is closer to the safe boundary of o_3. Thus, the approximate collision constraint for r_i at $\boldsymbol{x}_i[k]$ can be described as:

$$A_3^T\boldsymbol{x}_i[k] \geq b_3 + \|A_3\|_2\rho.$$

The region is shown in Fig. 4.7 with the light blue region.

For the constraints that have been added at previous iterations, the approximated convex feasible region is always non-empty. However, for the new added constraints at iteration $m+1$, the resulting convex feasible region may be empty based the above approximation. For example, consider the situation shown in Fig. 4.8. Suppose the current iteration is $m+1$, and the first position whose value violates some constraints is



(a) No feasible approximate convex region for $\boldsymbol{x}_i[k']$. (b) The approximate convex region for $\boldsymbol{x}_i[k']$.

FIG. 4.8: Convexification of a new added kind of collision avoidance constraints.

$\boldsymbol{x}_i[k']$, where $k' \in \mathcal{H}_i[k_0]$. Fig. 4.8(a) shows the convex approximation based on (4.14) with the value ${}^m\boldsymbol{x}_i[k']$. The region above l_1 is the collision-free region with respect to r_{j_1}, and the region below l_2 is the collision-free region with respect to r_{j_2}. Since l_1 is parallel with l_2, these two collision-free regions have no intersection. Thus, the feasible region of the approximate problem at the current iteration is empty. A possible way to avoid such a situation is to convexify the new non-convex constraints using the last position that does not cause any collisions with other robots or obstacles, rather than ${}^m\boldsymbol{x}_i[k']$. For example, Fig. 4.8(b) shows an approximation with the value ${}^m\boldsymbol{x}_i[k'-1]$. Clearly, the convex approximation for this new added collision avoidance constraints can be described in (4.16) and (4.17).

$$\forall r_j \in \mathcal{O}_i^{\beta}[k_0], \forall o \in \mathcal{O}_i^{\alpha_1}[k_0],$$

$$(\boldsymbol{x}_i[k'] - \widetilde{\boldsymbol{x}}_j[k'])^T \frac{({}^m\boldsymbol{x}_i[k'-1] - \widetilde{\boldsymbol{x}}_j[k'])}{\|{}^m\boldsymbol{x}_i[k'-1] - \widetilde{\boldsymbol{x}}_j[k']\|_2} \geq 2\rho, \tag{4.16}$$

$$(\boldsymbol{x}_i[k'] - \boldsymbol{x}_o[k_0])^T \frac{({}^m\boldsymbol{x}_i[k'-1] - \boldsymbol{x}_o[k_0])}{\|{}^m\boldsymbol{x}_i[k'-1] - \boldsymbol{x}_o[k_0]\|_2} \geq \rho + \rho_o. \tag{4.17}$$

4.4.2 The Distributed Algorithm to Trajectory Planning

Applying (4.14)–(4.17), we can approximate all the non-convex constraints to be convex. Thus, the local optimization problem $P_i[k_0]$ is approximated to be a convex one, denoted as $CVX\_P_i[k_0]$. The approximate convex problem can be efficiently solved by some existing available software, such as CVX [199]. Algorithm 1 shows the detailed algorithm for r_i to solve the optimization subproblem at k_0.

Algorithm 1: iSCP for robot r_i at time instant k_0: $iSCP(r_i, k_0, P_i[k_0])$.

Input : Current position $\boldsymbol{x}_i[k_0]$ and velocity $\boldsymbol{v}_i[k_0]$, target position $\boldsymbol{x}_i^f$, prediction horizon H_i, detected obstacles $\mathcal{O}_i^{\alpha}[k_0]$, observed robots $\mathcal{O}_i^{\beta}[k_0]$ and their current positions, the accuracy ϵ, and maximal iterations $m_{\max}$.

Output: The predicted acceleration $\boldsymbol{a}_i[k_0]$, velocity $\boldsymbol{v}_i[k_0+1]$, and position $\boldsymbol{x}_i[k_0+1]$.

Initialization: initial values ${}^0\boldsymbol{x}_i(k_0+1:T_i)$; $m=0$; and $PosiIndex = \emptyset$;
while $m \le m_{\max}$ **do**
  $CurConstr = \emptyset$, $HasAdded = false$;
  for $k = k_0+1$ **to** k_{0,H_i} **do**
    if $k \in PosiIndex$ **then**
      `/* Convexify the constraints which have been considered at the previous iterations. */`
      $CVXConstr = ApproxCVX({}^m\boldsymbol{x}_i[k], \mathcal{O})$;
      $CurConstr = CurConstr \cup CVXConstr$;
    else if $(!HasAdded) \wedge ({}^m\boldsymbol{x}_i[k] \in \mathcal{O})$ **then**
      `/* New constraints with respect to` $\boldsymbol{x}_i[k]$ `are added. */`
      $HasAdded = true$;
      $PosiIndex = PosiIndex \cup \{k\}$;
      $CVXConstr = ApproxCVX({}^m\boldsymbol{x}_i[k'-1], \mathcal{O})$;
      `/* Convexify the constraints with` ${}^m\boldsymbol{x}_i[k'-1]$ `(`${}^m\boldsymbol{x}_i[k'-1] \notin \mathcal{O}$`), instead of` ${}^m\boldsymbol{x}_i[k]$`. */`
      $CurConstr = CurConstr \cup CVXConstr$;
  $CVX\_P_i[k_0] = Approx(P_i[k_0], CurConstr)$;
  $(\boldsymbol{aa}, \boldsymbol{vv}, \boldsymbol{xx}) = Solve(CVX\_P_i[k_0])$;
  ${}^{m+1}\boldsymbol{x}_i[k_0+1:T_i] = \boldsymbol{xx}$;
  if $\max_k \|{}^{m+1}\boldsymbol{x}_i[k] - {}^m\boldsymbol{x}_i[k]\|_\infty \le \epsilon$ **then**
    $\boldsymbol{a}_i[k_0] = \boldsymbol{aa}[1]$, $\boldsymbol{v}_i[k_0+1] = \boldsymbol{vv}[1]$, and $\boldsymbol{x}_i[k_0+1] = \boldsymbol{xx}[1]$;
    return;
  $m = m+1$;
if $m > m_{\max}$ **then**
  Report false and act some emergent actions, such as an emergency brake.

In this algorithm, the variable $PosiIndex$ collects the indexes of positions whose corresponding collision avoidance constraints have been taken into consideration at the previous iterations; $CurConstr$ is the set of the convex approximations of the constraints that have been added into the problem; $HasAdded$ is a boolean variable that denotes whether a new position is added into consideration at the current iteration: If yes, its value is true. The index of a new violated position at the current iteration can

---

Algorithm 2: The distributed trajectory planning approach for robot r_i.

Input : Current time instant k_0 and the current position $\boldsymbol{x}_i[k_0]$ and velocity $\boldsymbol{v}_i[k_0]$, target position $\boldsymbol{x}_i^f$, discrete time step h and the final time instant T, and the prediction horizon H_i, and the sensing range L.

Output: Robot r_i moves to its target without causing collisions.

1 Detect the current local environment, and determine the sets of obstacles $\mathcal{O}_i^\alpha$ and robots $\mathcal{O}_i^\beta$ within its sensing range;
2 Locate the obstacles;
3 Communicate with the robots in $\mathcal{O}_i^\beta$ and determine their current positions;
4 Construct the distributed optimization sub-problem P_i;
5 Call $iSCP(r_i, k_0, P_i)$ (i.e., Algorithm 1) to solve P_i ;
6 Return $\boldsymbol{a}_i[k_0]$, $\boldsymbol{v}_i[k_0+1]$, and $\boldsymbol{x}_i[k_0+1]$;
7 Actuate the robot with the control signal $\boldsymbol{a}_i[k_0]$;
8 Update the current time and state;

---

be added to $PosiIndex$ only when $HasAdded = false$, i.e., the condition described in Line 8. Once a new violated position is detected, $HasAdded = true$, i.e., Line 9. $\mathcal{O} = \mathcal{O}_i^\alpha[k_0] \cup \mathcal{O}_i^\beta[k_0]$ is the observed obstacles at the current time instant k_0; the function $ApproxCVX(^m\boldsymbol{x}_i[k], \mathcal{O})$ in Line 6 is used to convexify the collision avoidance constraints with the value $^m\boldsymbol{x}_i[k]$ based on (4.14) and (4.15), while that in Line 11 is used to convexify the new added collision avoidance constraints with the value $^m\boldsymbol{x}_i[k'-1]$ based on (4.16) and (4.17). The approximate convex problem is denoted as $CVX\_P_i[k_0]$. The function $Solve(CVX\_P_i[k_0])$ solves the approximate convex problem and returns the sequences of accelerations $\boldsymbol{aa}$, velocities $\boldsymbol{vv}$, and positions $\boldsymbol{xx}$, where $\boldsymbol{aa}$, $\boldsymbol{vv}$, and $\boldsymbol{xx}$ are the predicted values of $\boldsymbol{a}_i[k_0 : T_i - 1]$, $\boldsymbol{v}[k_0+1 : T_i]$, and $\boldsymbol{x}_i[k_0+1 : T_i]$ in the current horizon, respectively.

At last, the fully distributed algorithm for real-time motion planning is shown in Algorithm 2. The main time cost of the algorithm is to solve the optimization problem, i.e., Line 5 in Algorithm 2. Based on Algorithm 1, to solve it is to solve a set of approximate convex problems. Fortunately, each convex optimization problem can be solved in polynomial time [196]. Thus, each robot at each time instant can predict its trajectory in polynomial time if any.

Each trajectory generated by the proposed method only guarantees collision avoidance at the discrete positions, rather than the whole one. But it can be resolved by increasing the given safe radius with a proper value [103]. Take a circular obstacle as

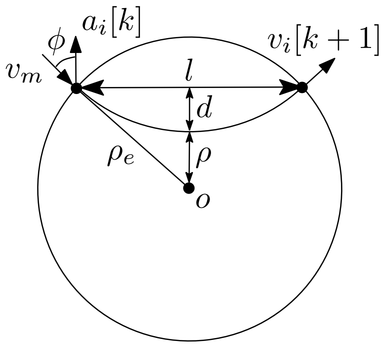


FIG. 4.9: The minimum distance between a robot and an obstacle in $[k, k+1]$.

an illustrative example. Indeed, at any time instant k, the acceleration between the instants k and $k+1$ is a constant. Thus, the minimum distance between the robot and an obstacle between k and $k+1$ happens when the two positions are at the boundary of the c-obstacle, which is shown in Fig. 4.9. In this case, the minimum distance between the robot and the obstacle is affected by the acceleration which is perpendicular to the line of the two successive positions of the robot. Thus, we have $v_m \cos\phi = a_i[k]h_i/2$, $d = h_i v_m \cos\phi/4$, and $l = h_i v_m \sin\phi$, where v_m is the maximum speed. Then,

$$\begin{aligned}
\rho_e^2 &= \frac{l^2}{4} + (\rho + d)^2 \\
&= \frac{h_1^2 v_m^2[k] \sin^2\phi}{4} + (\rho + \frac{h_i v_m \cos\phi}{4})^2 \\
&= -\frac{3}{16} h_i^2 v_m^2 \cos^2\phi + \frac{\rho h_i v_m}{2}\cos\phi + \rho^2 + \frac{1}{4} h_i^2 v_m^2 \\
&= -\frac{3}{16}(h_i v_m \cos\phi - \frac{4}{3}\rho)^2 + \frac{4}{3}\rho^2 + \frac{1}{4} h_i^2 v_m^2
\end{aligned}$$

Clearly, $\rho_e \leq \sqrt{\frac{4}{3}\rho^2 + \frac{1}{4}h_i^2 v_m^2}$. This can give us guidance to set the safe radius.

4.5 Simulated Cases: Implementation and Results

In this section, we use two examples to demonstrate the proposed algorithm. Without loss of generality, both are considered in a 2D space. In the first situation, a robot is moving in a dynamic environment where an obstacle will be placed and removed unexpectedly. The robot does not know in advance the global environment, as well as the information of the occurrence of obstacles. Thus, for the robot, the environment

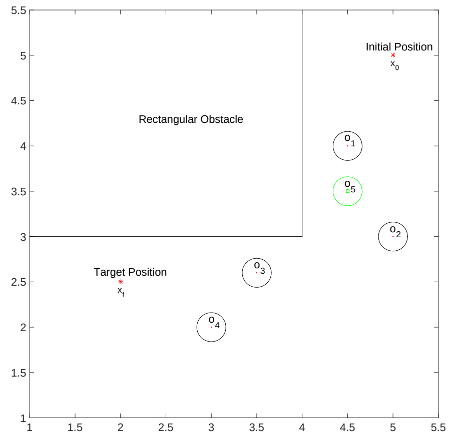


FIG. 4.10: The environment of Case 1. There are four static obstacles, i.e., $o_1 - o_4$, one removable obstacle o_5, and one rectangular obstacle. Each circle represents the corresponding c-obstacle.

is dynamic and unpredictable. In the second situation, a system with four robots is studied. In this scenario, each robot regards others as dynamic obstacles. In this sense, the environment is also a changing one. In our experiments, the convex problems are solved by CVX 2.1 with the default solver SDPT3 [199].

4.5.1 Case 1: One Robot in a Multi-Obstacle Environment

In this case, the parameters and running environment are set as follows. The time interval is discretized into $T = 30$ time instants with a time step $h = 0.1$; the prediction horizon is set as $H = 10$; and the sensing range is set as $L = 0.65$. The safe radius of each robot in the c-configuration space is $\rho = 0.16$, and the maximum velocity is $v_{\max} = 5$. However, at the computation phase, the safe radius is enlarged to 0.2 to avoid collisions on the trajectory segments between every two successive time instants. The environment is shown in Fig. 4.10. x_0 and x_f are the initial and target positions, respectively. Their coordinates are $(5, 5)$ and $(2, 2.5)$. The static obstacles in the environment are four circular obstacles and one rectangular obstacle. The corner of the rectangular obstacle is $(4, 3)$ and is a known obstacle at the beginning; while the coordinates of the circular obstacles are shown in Table 4.2. Obstacle o_5 is an obstacle that is placed to $(4.5, 3.5)$ at time instant $k_0 = 8$ and is taken away at $k_0 = 11$.

TABLE 4.2: Obstacle Positions in the Environment

Obstacle	o_1	o_2	o_3	o_4	o_5
Position	$(4.5, 4)$	$(5, 3)$	$(3.5, 2.6)$	$(3, 2)$	$(4.5, 3.5)$

TABLE 4.3: Obstacles that Are Detected at Different Time Instants

Time Instant(s)	Obstacle(s)
6, 7	o_1
8, 9, 10	o_1, o_5
11	o_1, o_2
12, 13, 14, 15,16	o_2
19, 20, 21, 22	o_3
23, 24	o_3, o_4
25, 26	o_4

Based on our method, at each discrete time instant, a robot builds a new optimization problem based on the current detected environment. Due to their similarity, it is not necessary to show the equations at all time instants. Rather, the equations at an arbitrary time instant are informative enough to represent others. Hence, we show the optimization problem at $k_0 = 8$, i.e., $P_i[8]$. Since there is only one robot, we omit the subscript index for the sake of clarity.

$$\min_{\boldsymbol{a}[8:29]} \sum_{k=8}^{17} \|\boldsymbol{a}[k]\|_2^2 + \sum_{k=18}^{29} \|\boldsymbol{a}[k]\|_2^2$$

subject to:

$$\boldsymbol{x}[k+1] = \boldsymbol{x}[k] + 0.1 * \boldsymbol{v}[k] + 0.01 * \frac{\boldsymbol{a}[k]}{2},$$

$$\boldsymbol{v}[k+1] = \boldsymbol{v}[k] + 0.1 * \boldsymbol{a}[k],$$

$$\|\boldsymbol{v}[k]\|_2 \in [0, 5], \|\boldsymbol{x}[j] - \boldsymbol{x}_{o_1}\|_2 \geq 0.2, \|\boldsymbol{x}[j] - \boldsymbol{x}_{o_5}\|_2 \geq 0.2,$$

$$\boldsymbol{x}[j] \notin \{(x, y) | x < 4 + 0.2, y > 3 - 0.2\};$$

$$\boldsymbol{x}[8] = (4.7086, 4.0467), \boldsymbol{v}[8] = (-0.6935, -2.0673),$$

$$\boldsymbol{x}[30] = (2, 2.5), \boldsymbol{v}[30] = (0, 0), \boldsymbol{x}_{o_1} = (4.5, 4), \boldsymbol{x}_{o_5} = (4.5, 3.5);$$

$$\forall k \in \{8, 9, \ldots, 29\}, \forall j \in \{9, 10, \ldots, 18\}.$$

To describe the affine approximation of the approved violated non-convex constrains at each iteration during the iSCP process, let $\boldsymbol{x}[j] = (x[j], y[j])$. Then the approximation of $\|\boldsymbol{x}[j] - \boldsymbol{x}_{o_1}\|_2 \geq 0.2$ at iteration m with ${}^m\boldsymbol{x}[j] = ({}^m x, {}^m y)$ can be described

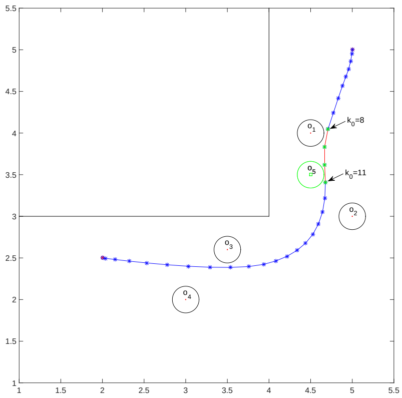


FIG. 4.11: The generated path. The red part is the path where obstacle o_5 is placed in the environment.

as:

$$({}^m x - 4.5)x[j] + ({}^m y - 4)y[j]$$
$$\geq 4.5 * ({}^m x - 4.5) + 4 * ({}^m y - 4) + 0.2 * \sqrt{({}^m x - 4.5)^2 + ({}^m y - 4)^2}.$$

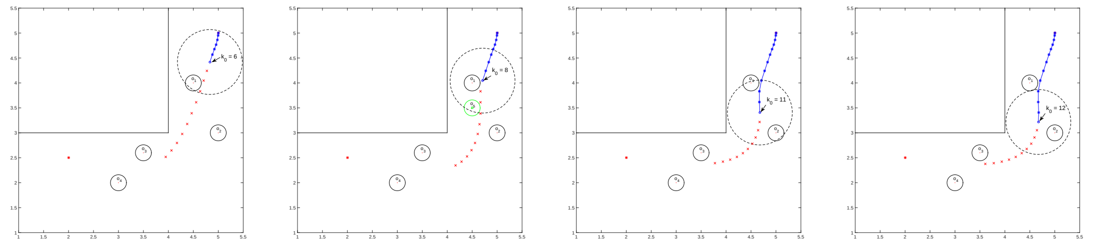
Similarly, the approximation of $\|\boldsymbol{x}[j] - \boldsymbol{x}_{o_5}\|_2 \geq 0.2$ can be described as:

$$({}^m x - 4.5)x[j] + ({}^m y - 3.5)y[j]$$
$$\geq 4.5 * ({}^m x - 4.5) + 3.5 * ({}^m y - 3.5) + 0.2 * \sqrt{({}^m x - 4.5)^2 + ({}^m y - 3.5)^2}.$$

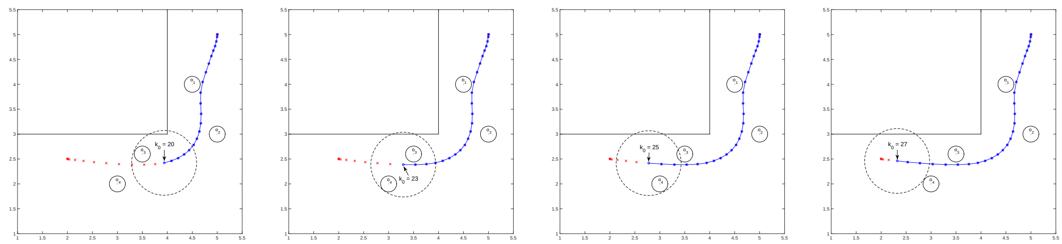
For the rectangular obstacle, the constraint $\boldsymbol{x}[j] \notin \{(x, y)|x < 4 + 0.2, y > 3 - 0.2\}$ is approximated as $y[j] \leq 2.8$.

Based on the above implementation, the traversed path of the robot is shown in Fig. 4.11, and the observed obstacles during the motion are shown in Table 4.3. In Fig. 4.11, the red part of the path is the path passed through with the existence of o_5. We can find that the robot moves away from the original orientation to the target because of the sudden observation of o_5 at the time instants $8-11$. Detailedly, during the move among these time instants, to avoid collision with o_5, the robot needs to make a turn, causing a jink on the path. When the obstacle o_5 disappears at $k_0 = 11$, the robot begins to move towards the target.

Fig. 4.12 shows some snapshots of the real-time trajectories at different time instants. As shown in Fig. 4.12(a), at $k_0 = 6$, there is only one obstacle o_1 within the robot's sensing range. So the local trajectory in the prediction horizon only needs to



(a) $k_0 = 6$ and $\mathcal{O}_i[k_0] = \{o_1\}$.
(b) $k_0 = 8$ and $\mathcal{O}_i[k_0] = \{o_1, o_5\}$.
(c) $k_0 = 11$ and $\mathcal{O}_i[k_0] = \{o_1, o_2\}$ with the move of o_5.
(d) $k_0 = 12$ and $\mathcal{O}_i[k_0] = \{o_2\}$.



(e) $k_0 = 20$ and $\mathcal{O}_i[k_0] = \{o_3\}$.
(f) $k_0 = 23$ and $\mathcal{O}_i[k_0] = \{o_3, o_4\}$.
(g) $k_0 = 25$ and $\mathcal{O}_i[k_0] = \{o_4\}$.
(h) $k_0 = 27$ and $\mathcal{O}_i[k_0] = \emptyset$.

FIG. 4.12: The traversed and predicted trajectory in different prediction horizon. The solid circles represent the forbidden regions of the corresponding circular obstacles $o_1 - o_4$. The dashed circle in each sub-figure represents the boundary of the sensor. The curves with star dots are the pathes that have been traversed by the robot, while the cross dots represent the predicted positions of the prediction horizon.

avoid collision with o_1. When $k_0 = 8$, a new obstacle o_5 is placed into its sensing range, so the current obstacles are o_1 and o_5 (also shown in Table 4.3). As shown in Fig. 4.12(b), this causes the trajectory to deviate from the original direction. Since o_5 is taken away at time instant 11, there are only two obstacles, o_1 and o_2, detected at $k_0 = 11$ and shown in Fig. 4.12(c). At $k_0 = 12$, obstacle o_1 is no longer in the range of the robot. So the observed obstacle is o_2, which is shown in Fig. 4.12(d). Note that the predicted trajectory is only required to avoid collisions the robot detects, so it is available even though the trajectory in the current horizon collides with another obstacle o_3. The same analysis can be done for Figs. 4.12(e)–4.12(h).

4.5.2 Case 2: Multiple Robots in an Obstacle-Free Environment

In this subsection, we consider the implementation of our approach in a multi-robot system which is executed in an obstacle-free environment. As shown in Fig. 4.13, there are four robots, say $r_1 - r_4$, in the system. Their initial positions are $(1, 1)$, $(6, 1)$, $(6, 6)$,

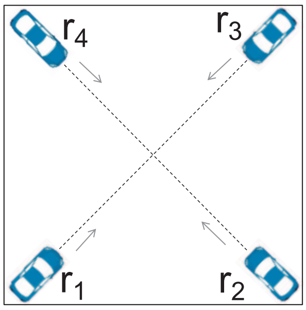


FIG. 4.13: A simulation multi-robot system with 4 robots. Robots r_1 and r_3 are required to exchange their positions, and r_2 and r_4 are required to exchange the positions.

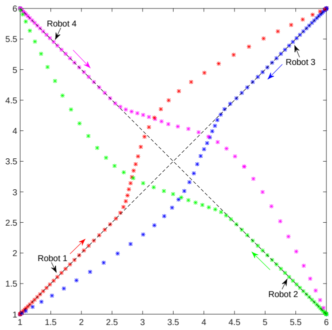
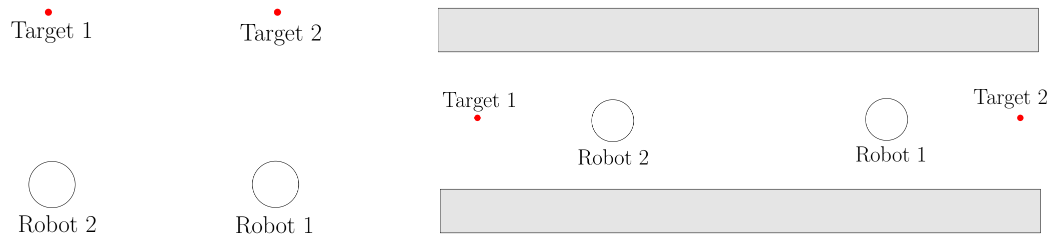


FIG. 4.14: The paths traversed by the four robots.

and $(1, 6)$, respectively. The tasks are that $r_1 - r_4$ need to move to the initial positions of r_3, r_4, r_1, and r_2, respectively.

The parameters of this case study are set as follows. $T_i = 60$, $H_i = 15$, $h_i = 0.1$, $\|\boldsymbol{v}_i\|_2 \leq \sqrt{3}$, where $i = 1, \ldots, 4$; $\rho = 0.35$, and $L = 2$. Note that each robot can start individually at any time. In our case, we suppose the four robots start sequentially but with very short delays. Fig. 4.14 shows the paths that the four robots traverse. At the beginning of its motion, each robot moves directly to its target since there are no obstacles detected. When the robots move towards others, they deviate from the original directions in order to avoid collisions with each other. From the paths, we can find that to pass through the intersection area without any collisions, each robot deviates to the left of its motion with proper negotiations. Indeed, such motion of these robots is analogous to the movement within a roundabout in the traffic systems.



(a) Example 1. (b) Example 2.

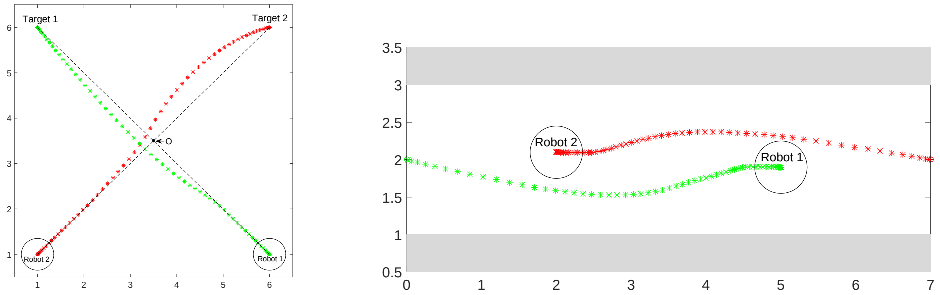
FIG. 4.15: Illustrative examples for livelock avoidance.

4.5.3 Case 3: Multiple Robots with Symmetric Trajectories

In our method, at each time instant, each robot will re-search for its neighbors and re-predict its neighbors' positions. Once a collision is found, a robot will plan a new collision-free trajectory. Thus, if two robots are moving directly to each other, they would change their directions in advance to avoid deadlocks. After it passes through the collision regions, a robot will plan the new trajectory to the target position.

Note that for a distributed method, we do not mean that there are no negotiations among robots. Indeed, for any decentralized or distributed method, negotiations are inevitable in order to avoid synchronous moves. In our method, robots may also need negotiations to avoid livelocks. A livelock is a situation that some robots keep moving but will never reach their targets. It may arise because all the configuration parameters of these robots are the same, and at each instant, they are computing the collision-free trajectories and updating positions simultaneously. This means that their trajectories are always symmetric. However, with negotiations, these robots can determine a solution to avoid livelocks. Note that the prediction methods of other robots' motion do not affect the negotiation process.

Consider two systems shown in Fig. 4.15. Assume that the safe radius is 0.35. In Fig. 4.15(a), the two robots are at $(6, 1)$ and $(1, 1)$. They need to move to $(1, 6)$ and $(6, 6)$, respectively. At the beginning, the trajectories are symmetric. When they are near the intersecting point O, they will negotiate with each other to avoid the simultaneous computation and execution for collision avoidance. Thus, the generated result is that they move in a coordinated manner. Their traversed paths are shown in Fig. 4.16(a). In Fig. 4.15(b), the initial positions of the two robots are $(5, 1.9)$ and $(2, 2.1)$, respectively;



(a) Paths for example 1. (b) Paths for example 2.

FIG. 4.16: The generated paths without any livelock.

the targets are $(0, 2)$ and $(7, 2)$, respectively. The boundaries of environment are $y \leq 3$ and $y \geq 1$. Similarly, the generated paths are shown in Fig. 4.16(b).

The video of the three simulations can be found at `https://youtu.be/8kYI_DueEH8`.

4.6 Discussion

This section gives more discussions on the proposed method.

First, the proposed method can deal with both holonomic and nonholonomic kinematics. The difference between holonomic and nonholonimic kinematics is that the holonomic kinematics only consider the constraints on positions, while the nonholonomic kinematics consider the constraints containing velocities and other derivatives of positions. Since we do not restrict the constraints in order to construct the optimization problem for each robot, our method can deal with different kinematics. Besides, the proposed method can be used in both 2D and 3D scenarios. The main difference is the number of control variables. Our method does not limit the number of variables. However, we must point out the number of variables will affect the computation cost.

Second, the proposed method is suitable for different environments, especially for dynamic environments and the environments without priori knowledge. Indeed, with the MPC strategy, each robot can update its detection information, based on which the robot can replan its trajectory to fit the new environment.

Third, the proposed method is suitable for the situations that each robot can freely plan its trajectory in the environment. However, by adding corresponding constraints, the proposed method can also be applied for the situation that some robots are required to move along some reference paths in some areas. Because of the classical physical laws, the generated path by each robot is a continuously differential curve. Thus, for partial differential reference paths, some preprocessing procedures may be needed to smooth the paths. For example, we can use a proper tangent arc of the path to smooth a non-differential point in the reference path.

At last, the proposed method focuses on the robots that can always work well. However, it is also suitable for the systems containing robots that may fail at any time. In this case, other robots can still plan their trajectories only by regarding the failed ones as static obstacles.

4.7 Conclusion

We, in this chapter, propose a real-time and fully distributed algorithm to plan trajectories for multi-robot systems without the priori knowledge of the environment. The proposed method is a combination of the MPC strategy, which is a general framework to solve problems real-timely in the time domain, and iSCP, which is an efficient method used to solve the non-convex optimization subproblems. To construct its optimization subproblem in each prediction horizon, a robot needs to know the positions of the obstacles and the current states of other robots within its sensing range. We also prove that such knowledge can be obtained with the minimal amount of communication. With the information of current states of its neighbors, a robot first predicts the possible future positions of others, and then builds constraints to avoid collisions with them. Since it predicts the motion of others by itself, a robot does not need to wait for the computation of other robots. Thus, each robot can compute its trajectories and update its position independently with the retrieved information. So the approach we proposed is fully distributed. The computation complexity of the proposed method is polynomial time.

Chapter 5

Discrete Modeling of Robot Motion in Multi-Robot Systems with Fixed Paths

In Chapter 4, we study a distributed approach to trajectory planning of multi-robot systems where each robot can change its path freely. When these paths are recorded, the future robots may move following these fixed paths due to similar objectives and tasks. Moreover, because of the limitations of the environment or infrastructure, a robot has to move along a predetermined path. In these scenarios, each robot in a multi-robot system has a predetermined path. Thus, not only collisions but also deadlocks may occur during robot motion. In the sequel, we focus on motion control of such systems. By assuming proper local continuous controllers are available for robots, we study motion control problems from the theory of discrete event systems (DESs). In this chapter, we first give the discrete model for the motion of such a system, and in the next three chapters, we study some detailed motion control problems based on this discrete model.

5.1 Introduction

Many the state-of-the-art motion planning methods are for multi-robot systems where robots are moving in a free environment and can change their paths freely if needed. However, in some scenarios, especially in transportation systems, warehouses, tourist areas, and public parks, a robot may have to move along a fixed path due to infrastructure limitations, task requirements, and so on. For example, different autonomous vehicles may be required to move along different lines to monitor the traffic conditions persistently; robots in warehouses are required to continually load and unload materials or products in the given lines; and cars in tourist areas run in circles to carry tourists. In these examples, robots are required to move along predetermined paths to perform given tasks. Moreover, with the state-of-the-art path planning algorithms, we may first obtain paths accommodating infrastructure limitation [17] and special task requirements [25, 200, 201], and then fix robots to move along these special paths. In these systems, we always need to make sure that there are no environmental obstacles on the paths.

Since it can be an arbitrary curve, sometimes the path of a robot cannot be described as a closed-form expression. As an alternative, discrete representation is an alternative to reduce the computational complexity significantly [54]. Indeed, to model the complex control system in a way that facilitates the analysis and verification of its performance, it is common to model a system at different levels of abstraction, from high-level discrete control to low-level continuous control [107,130,202]. For example, in [130], by abstracting disjoint collision zones, Soltero *et al* study collision and deadlock avoidance in multi-robot systems where robots have intersecting paths. Then some optimal policies are designed to determine the sequence of robots entering a collision zone. Reveliotis *et al* [162, 203] study the motion planning problem from the resource allocation paradigm, where the motion space is discretized into a set of cells. Regarding these cells as resources, each robot decides which resources it needs at different zones. In this chapter, using labeled transition system (LTS) model, we specify formally the motion of a robot along a given path by discretizing its path. In the rest of this chapter, Section 5.2 describes the determination of collision segments of a path; Sections 5.3 and 5.4 describe the building of the discrete state space of a robot and its LTS model, respectively; and finally Section 5.5 gives some discussion on the discrete abstraction of robot motion.

5.2 Determination of Collision Segments

Given a system with N robots, suppose each robot has a predetermined and closed path $p^i(\theta)$ (sometimes simply denoted as p^i), $i \in \mathbb{I}_N = \{1, 2, \ldots, N\}$. A segment of p^i is a continuous part of the path p^i and can be described as $p^i_k = \{p^i(\theta)|\theta \in [\theta_1, \theta_2], \theta_1, \theta_2 \in [0, 1]\}$. $p^i(\theta_1)$ is called tail of p^i_k and $p^i(\theta_2)$ is head of p^i_k. Given two segments p^i_k and $p^j_{k'}$, their distance can be computed as $d(p^i_k, p^j_{k'}) = \inf_{x \in p^i_k, y \in p^j_{k'}} d(x, y)$.

Definition 6 (Robot Motion). The motion of r_i along p^i is a binary relation $\rightarrow_{p^i}$ on p^i, i.e., $\rightarrow_{p^i} \subset p^i \times p^i$: $\forall x, y \in p^i$, $(x, y) \in \rightarrow_{p^i}$, denoted as $x \rightarrow_{p^i} y$, if r_i can move from x to y along p^i.

Since p^i is a closed path and r_i is doing persist motion, we have:

- $\forall x \in p^i$, $(x, x) \in \rightarrow_{p^i}$. This means that for any position x on p^i, r_i can move back to x, i.e., r_i moves along p^i for one round.
- $(x, y) \in \rightarrow_{p^i} \Rightarrow (y, x) \in \rightarrow_{p^i}$. It means that if a robot can move from x to y, then it can move from y to x. Their traversed paths form the whole p^i.
- $(x, y) \in \rightarrow_{p^i}$ and $(y, z) \in \rightarrow_{p^i} \Rightarrow (x, z) \in \rightarrow_{p^i}$. This means if r_i can move from x to y, and y to z, then it can definitely move from x to z.

For the sake of safety, each robot has a safe radius, say ρ, during its motion. In terms of safe radius, the real footprints of a robot can be regarded as a sphere with a radius ρ. Thus, two robots are in a collision if $d(x_0, y_0) < 2\rho$, where x_0 and y_0 are their positions. Hence, the safe region for r_i's motion at position x_0 is sphere $\|x - x_0\|_2 \leq 2\rho$, meaning that other robots cannot be in this sphere at that time; the whole safe region for r_i can be described as $A^i_{2\rho} = \{x \in \mathbb{R}^{n_0} | \|x - p^i(\theta)\|_2 \leq 2\rho, \theta \in [0, 1]\}$. Clearly, r_i's collision locations with r_j is $p^i(j) = p^i \cap A^j_{2\rho}$.

For example, Fig. 5.1 shows an example of safe regions of robots in a 2D case. The two solid circles, whose radii are ρ, show the footprints of robots in term of safe radius. The dashed circle with a radius 2ρ is the safe region of r_i when it is at x_0. Other robots, such as robot r, cannot move into this region at that time; otherwise, a collision occurs due to the intersecting of their footprints. The red and blue solid curves are segments of

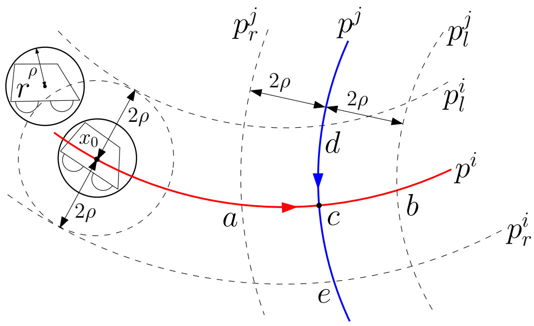


FIG. 5.1: An example to show safe regions of robots in 2D motion space. Solid circles at the left are the real motion space with a safe radius ρ, the dashed circle at x_0 is the safe region of r_i when it is at x. Solid curves p^i and p^j are the paths of r_i and r_j. Their safe regions are bounded by the parallel boundaries $\langle p_l^i, p_r^i\rangle$ and $\langle p_l^j, p_r^j\rangle$.

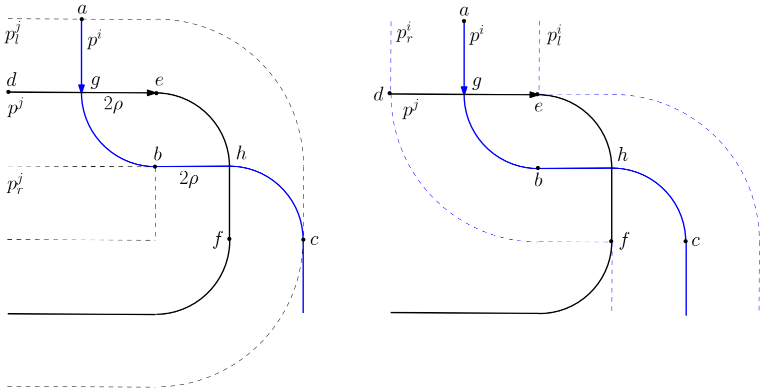
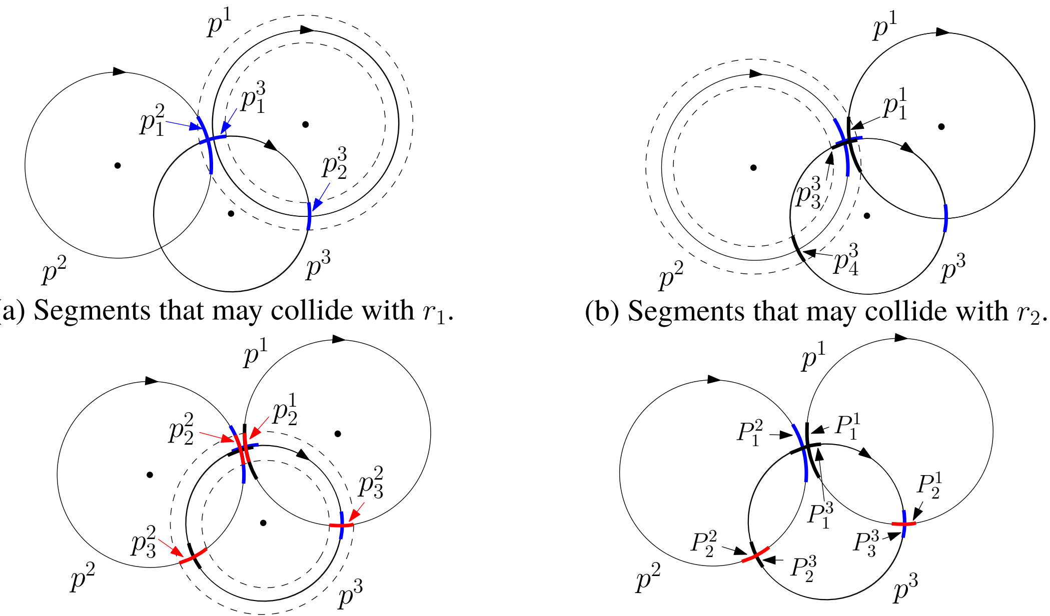


FIG. 5.2: An example to illustrate the maximal continuous segments.

p^i and p^j, respectively. The dashed curves p_l^i and p_r^i are the boundaries of $A_{2\rho}^i$, and p_l^j and p_r^j are boundaries of $A_{2\rho}^j$. Hence, the segment $\widehat{acb}$ is a segment in $p^i(j)$ and $\widehat{dce}$ is in $p^j(i)$. Here a and b are tail and head of $\widehat{acb}$, respectively. Note that each sub-segment of $\widehat{acb}$ is also a segment in $p^i(j)$.

Definition 7. A segment p_k^i is called a *collision segment* with r_j on p^i if p_k^i is a maximal continuous segment in $p^i(j)$. The set of collision segments with r_j is denoted as $P^{i,j}$.

Remark 3. Note that the segments in $p^i(j)$ is infinite. So, in the above definition, we only consider the maximal continuous segments. For example, as shown in Fig. 5.2, even though $\widehat{agbhc}$ can be divided into two parts $\widehat{agb}$ and $\widehat{bhc}$, such that $\widehat{agb}$ may cause collisions only when r_j is at $\widehat{dge}$ and $\widehat{bhc}$ may cause collisions only when r_j is at $\widehat{ehf}$, one of the collision segments with r_j on p^i is $\widehat{agbhc}$ but not $\widehat{agb}$ and $\widehat{bhc}$. Similarly, $\widehat{dgehf}$ is a collision segment with r_i on p^j.



(a) Segments that may collide with r_1.

(b) Segments that may collide with r_2.

(c) Segments that may collide with r_3.

(d) Final collision segments.

FIG. 5.3: An example to show collisions among multiple robots.

Fig. 5.3 shows an example of collision segments among three robots in a multi-robot system. Similarly, the solid circles denote robots' paths p^i, and the dashed circles denote the boundaries of $A^i_{2\rho}$. Fig. 5.3(a) shows r_2's and r_3's collision segments, i.e., the blue bold segments, with r_1. Fig. 5.3(b) shows the collision segments of r_1 and r_3 with r_2, i.e., the black bold segments; and Fig. 5.3(c) shows the collision segments of r_1 and r_2 with r_3, i.e., the red bold segments. Note that in Fig. 5.3(b), the overlapped segment of the blue and black bold segments on p^3, i.e., p^3_1 and p^3_3, means that r_3 may collide with r_1 and r_2 simultaneously if it is at this part, so do the overlapped parts of p^2_1 and p^2_2, and p^1_1 and p^1_2.

Indeed, by searching for its path, each robot can determine its collision segments with others independently. First, using its vision and distance sensors, a robot can detect the paths of other robots. For each path ahead, the robot can determine its maximal continuous segment such that the minimal distance of each point to the detected path is less than 2ρ. For example, as shown in Fig. 5.4, searching for its path p^2, r_2 detects that a is the first point from which the distance to p^1 is less than 2ρ, and b is the last point whose distance to p^1 is less than 2ρ. Thus, segment $\widehat{ab}$, the bold blue curve, is a collision segment with r_1 on p^2. Similarly, $\widehat{cd}$ is a collision segment with r_3 on p^2.

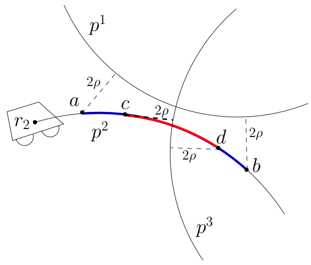


FIG. 5.4: An example to show the detection of collision segments by a robot.

Once a robot determines its collision segments with other robots, to simplify the analysis, it merges the overlapped segments. Thus, the set of the *final non-overlapped collision segments* is $P^i = \cup_j P^{i,j} \triangleq \{P^i_1, P^i_2, \ldots\}$ with overlapped merging. For example, p^3_1 and p^3_3 in Fig. 5.3 are merged into P^3_1, shown in Fig. 5.3(d); p^2_1 and p^2_2 are merged into P^2_1; and p^1_1 and p^1_2 are merged into P^1_1. Hence, as shown in Fig. 5.3(d), after merging, the collision segments of r_1, r_2, and r_3 are $\{P^1_1, P^1_2\}$, $\{P^2_1, P^2_2\}$, and $\{P^3_1, P^3_2, P^3_3\}$, respectively. In the following, without ambiguity, a collision segment means a final non-overlapped collision segment.

Definition 8. $(P^i_k, P^j_{k'})$ is a collision pair between p^i and p^j if P^i_k and $P^j_{k'}$ are their collision segments, and $d(P^i_k, P^j_{k'}) < 2\rho$.

For example, as the example shown in Fig. 5.3(d), their collision pairs are: (P^2_1, P^1_1), (P^3_1, P^2_1), (P^1_1, P^3_1), (P^3_3, P^1_2), and (P^3_2, P^2_2).

Proposition 1. Collisions between r_i and r_j can only occur in their collision pairs.

Proof. Suppose r_i and r_j are in a collision when they are at x_0 and y_0. Then there exist P^i_k and $P^j_{k'}$ such that $x_0 \in P^i_k$ and $y_0 \in P^j_{k'}$. Since $d(P^i_k, P^j_{k'}) \le d(x_0, y_0) < 2\rho$, $(P^i_k, P^j_{k'})$ is a collision pair. □

5.3 Abstraction of Discrete States

In this section, we describe the process to build the discrete state space of each robot. Consider robot r_i's path p^i.

When collision segments are determined, the rest of p^i, i.e., $p^i \setminus P^i$, is always collision-free. Each maximal continuous segment in $p^i \setminus P^i$ may be further partitioned

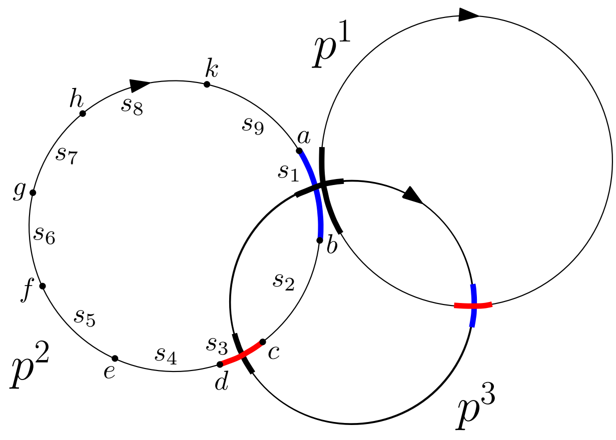


FIG. 5.5: An example to show discretization of a path.

into a set of smaller segments based on some parameters such as sensing ranges. Such a smaller segment is called a *private segment*. In this way, p^i is finally partitioned into a set of segments, which can be classified into two kinds: collision segments in P^i and private segments in $p^i \setminus P^i$. We denote such segments as $P^i_d = \{P^i_k,\ k = 1,\ 2,\ \ldots,\ n^i\}$. Here we assume that each P^i_k includes its head but excludes its tail, i.e., $P^i_k = \{p^i(\theta)|\theta \in (\theta_{k,1}, \theta_{k,2}]\}$. So $P^i_{k_1} \cap P^i_{k_2} = \emptyset$ for $k_1 \neq k_2$. For example, as shown in Fig. 5.5, the collision segments are $(a, b]$ and $(c, d]$. Here the notation $(a, b]$ means the directed arc $\widehat{ab}$ excluding a but including b. The directed arc $(d, a]$ is divided into a set of smaller segments: $(d, e]$, $(e, f]$, $(f, g]$, $(g, h]$, $(h, k]$, and $(k, a]$. Hence, we have $P^2_d = \{(a, b], (b, c], (c, d], (d, e], (e, f], (f, g], (g, h], (h, k], (k, a]\}$.

Based on above segments, we first define a binary relation $\equiv_d$:

Definition 9. For any two points x and y on p^i, $x \equiv_d y$ if and only if $\exists k$ such that $x \in P^i_k$ and $y \in P^i_k$.

Clearly, $\equiv_d$ is an equivalence relation and each P^i_k, $k = 1, 2, \ldots, n^i$, is an equivalence class. So all P^i_k form a partition of p^i. By abstracting each equivalence class P^i_k as a discrete state s^i_k, we can obtain a discrete state space of r_i, denoted as S^i. Note that the segments in a collision pair $(P^i_k, P^j_{k'})$ are defined as the same discrete state. For example, as shown in Fig. 5.3(d), P^1_1, P^2_1, and P^3_1 are abstracted as the same discrete state. In the discrete form, we say a robot is at a state s^i_k if its current position $x_0 \in P^i_k$. Hence, S^i can be classified into collision states S^i_α, which are from segments in P^i, and private states S^i_β, which are from segments in $p^i \setminus P^i$. For example, the discretization of p^2 shown in Fig. 5.5 contains 9 equivalence classes, i.e., the segments in P^2_d. Each is then abstracted to a discrete state. Hence $S^2 = \{s_1, s_2, \ldots, s_9\}$, where s_1 corresponds to $(a, b]$, s_2 to $(b, c]$, s_3 to $(c, d]$, …, s_9 to $(k, a]$. Clearly, $S^2_\alpha = \{s_1, s_3\}$ and $S^2_\beta = \{s_2, s_4, \ldots, s_9\}$.

Next, we state the relation between collision in terms of continuous path and discrete states. Let $f^i : p^i \rightarrow S^i$ be a function mapping from continuous positions in p^i to discrete states in S^i such that $\forall x \in P^i_k$, $f^i(x) = s^i_k$.

Theorem 2. Given two robots r_i and r_j, if they are in a collision at x_0 and y_0, then they are at the same state.

Proof. If r_i and r_j are in a collision when they are at x_0 and y_0, then based on Proposition 1, there exists a collision pair $(P^i_k, P^j_{k'})$ such that $x_0 \in P^i_k$ and $y_0 \in P^j_{k'}$. Based on the abstraction of discrete states, P^i_k and $P^j_{k'}$ are abstracted to the same discrete state s, so $f^i(x_0) = s$ and $f^j(y_0) = s$. Hence, r_i and r_j at the same state. □

This theorem means that such discritization does not lose any collision. Two robots cannot cause any collision if they are at different states. But we must point out that if two robots are at the same state, they may not collide with each other.

5.4 Labeled Transition Systems Modeling

Based on the discrete states determined in the above section, we can build the detailed LTS model for each robot.

First, the finite set of states of robot r_i is S^i. For convenience, let $S^i = \{s^i_k : k = 1, 2, \ldots, n_i\}$.

Second, consider the set of events. In terms of the discrete states, a robot can move from one state to another due to change of its positions. A robot may also need to stop its motion in order to avoid collisions. Hence, there are two actions: move and stop. This means $\Sigma_i = \{move, stop\}$.

Third, consider the set of transitions $\rightarrow_i$ for r_i. On one hand, for each state $s^i_k \in S^i$, r_i move to a different state as the robot is doing persistent motion. Since its motion is predetermined, r_i can only move to a determined state. Therefore, there exists a unique state $s^i_{k'}$ such that $s^i_k \xrightarrow{move}_i s^i_{k'}$. This kind of transitions is denoted as $\rightarrow_{i,move} = \{s^i_k \xrightarrow{move}_i s^i_{k'} : k = 1, 2, \ldots, n_i$, and $s^i_{k'}$ is uniquely determined by s^i_k. In fact, the determination of $s^i_{k'}$ can be described as follows. Suppose s^i_k is the discrete state of P^i_k. Along the motion direction, suppose the first segment connecting to P^i_k is $P^i_{k'}$, whose discrete state is $s^i_{k'}$. Thus, we have $s^i_k \xrightarrow{move}_i s^i_{k'}$. For example, as shown in Fig. 5.5, the first segment connecting to $(a, b]$ along its motion direction is $(b, c]$, and so we have $s_1 \xrightarrow{move}_i s_2$. On the other hand, robot r_i can stop at any state s^i_k. Thus, there is another transition for each s^i_k, i.e., $s^i_k \xrightarrow{stop}_i s^i_k$. The set of all this kind of transitions is denoted as $\rightarrow_{i,stop} = \{s^i_k \xrightarrow{stop}_i s^i_k : \forall s^i_k \in S^i\}$.

Hence, the detailed LTS model for robot r_i is

$$\mathcal{T}_i = \langle S^i, \Sigma_i = \{move, stop\}, \rightarrow_i\rangle \tag{5.1}$$

where $S^i = S^i_\alpha \cup S^i_\beta$ and $\rightarrow_i = \rightarrow_{i,move} \cup \rightarrow_{i,stop}$.

Remark 4. Note that the LTS model describes robot motion from the high-level discrete abstraction by considering two simple actions: $move$ and $stop$. Compared to the low-level continuous motion, the discrete transitions can be triggered at the current state only when a robot reaches the head of the corresponding segment.

Let $Pre_i(s) = \{s' \in S^i | s' \xrightarrow{move}_i s\}$ and $Pos_i(s) = \{s' \in S^i | s \xrightarrow{move}_i s'\}$. Based on the discrete model, $\forall s \in S^i$, $|Pre_i(s)| = |Pre_i(s)| = 1$. Thus, for convenience, throughout the thesis, we use $Pre_i(s)$ and $Pre_i(s)$ to denote the unique preceding and succeeding states of s in S^i, respectively.

Based on the LTS models of the robots in a system, we can give the LTS model of the whole system.

Definition 10. Let $\mathcal{T}_i = \langle S^i, \Sigma_i, \rightarrow_i\rangle$ be the LTS model of robot r_i, $i \in \mathbb{I}_N$. The entire system can be described as the parallel composition of all the individual transition systems, i.e., $\mathcal{T} = \mathcal{T}_1 || \cdots || \mathcal{T}_N = \langle C, \Sigma, \rightarrow\rangle$, where

1. $C = S^1 \times \ldots \times S^N$;

2. $\Sigma = \cup\, \Sigma_i$ is the set of labels;

3. $\rightarrow = \cup_{i\in\mathbb{I}_N} \rightarrow_i$ is the set of transitions, $\forall c_1 = (s_1^1, s_1^2, \ldots, s_1^N) \in C$, $c_2 = (s_2^1, s_2^2, \ldots, s_2^N) \in C$, $(c_1, c_2) \in \rightarrow_i$ if $(s_1^i, s_2^i) \in \rightarrow_i$, while $s_1^j = s_2^j$ for $j \neq i$.

In an arbitrary configuration $c = (s^1, s^2, \ldots, s^N)$, $c(i) = s^i$ is the state of robot r_i. The set of all collision states is $S_\alpha = \cup_{i\in\mathbb{I}_N} S_\alpha^i$.

Proposition 2. In a multi-robot system $\mathcal{T}$, for any state s_1 and s_2, there exists at most one robot r_i such that $s_1 \xrightarrow{move}_i s_2$.

Proof. Suppose there are two robots r_i and r_j, and two states s_1 and s_2 such that $s_1 \xrightarrow{move}_i s_2$ and $s_1 \xrightarrow{move}_j s_2$. The collision pairs of s_1 and s_2 are $(P_{k_1}^i, P_{k_1'}^j)$ and $(P_{k_2}^i, P_{k_2'}^j)$, respectively. Hence, $P_{k_1}^i$ and $P_{k_2}^i$ are two collision segments with r_j. Since $s_1 \xrightarrow{move}_i s_2$, $P_{k_1}^i$ and $P_{k_2}^i$ are successive. However, based on Definition 7, a collision segment is the maximal continuous segment, $P_{k_1}^i$ and $P_{k_2}^i$ should be one segment. So do $P_{k_1'}^j$ and $P_{k_2'}^j$. Hence s_1 and s_2 should be one state. $\square$

This proposition means that two successive collision states of a robot must collide with two different robots. For example, in Fig. 5.2, r_i and r_j may collide with each robot only when they are at $\widehat{agb}$ and $\widehat{dge}$ at the same time, or at $\widehat{bhc}$ and $\widehat{ehf}$ simultaneously. However, since $\widehat{agb}$ and $\widehat{bhc}$ are successive and may collide with the same robot r_j. So $\widehat{agbhc}$ is a collision segment, rather than $\widehat{agb}$ or $\widehat{bhc}$.

In the graphic representation of the LTS model of a multi-robot system, each circle represents a state. For the sake of simplicity, we do not explicitly show the self-loop transitions and labels in the graphic representation of LTS models. A state with a number i denotes the current state of r_i. Arcs with the same color represent the transitions of a robot. Different colors represent different robots and their transitions. For example, Fig. 5.6 shows a part of the LTS model of a system containing r_i, r_j, and r_k. The current states of r_i, r_j, and r_k are s_1, s_5, and s_7, respectively. The black arcs are the $move$ transitions of r_i, while the red ones are $move$ transitions of r_j, and the blue ones are $move$ transitions of r_k. The transitions of r_j among the given three states are $s_5 \xrightarrow{move}_j s_2$, $s_2 \xrightarrow{move}_j s_6$, $s_5 \xrightarrow{stop}_j s_5$, $s_2 \xrightarrow{stop}_j s_2$, and $s_6 \xrightarrow{stop}_j s_6$.

At last, we give some assumptions based the discrete model.

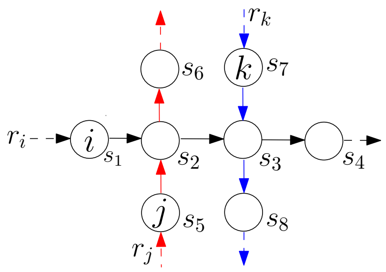


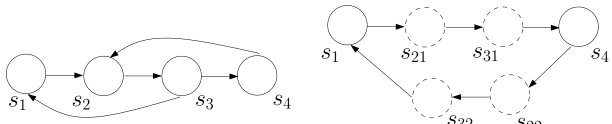
FIG. 5.6: A part of the LTS model of a multi-robot system containing three robots r_i, r_j, and r_k. The current states of r_i, r_j, and r_k are s_1, s_5, and s_7, respectively.

1. Path Assumptions. Each path is a one-way traffic. This means each robot is not allowed to move back. Thus, for any two states s_1 and s_2 satisfying $s_1 \xrightarrow{move}_i s_2$, the transition $s_2 \xrightarrow{move}_i s_1$ is forbidden. Besides, $S^i_\beta \neq \emptyset$ and $S^i_\alpha \neq \emptyset$. Indeed, if $S^i_\beta = \emptyset$, the task of r_i can be finished by other robots. Thus, the system can be refined by removing r_i. If $S^i_\alpha = \emptyset$, r_i can always move independently.

2. To avoid conflicts during simultaneous motion, a robot needs to identify the robots that are moving to a same state/region at the same time, and negotiate with them to determine the one that can fire its current move transition. Physically, all robots can move along their continuous paths simultaneously.

5.5 Discussion and Conclusion

In this chapter, we describe the process to build the LTS model for each robot. The main task is to discretize the path of a robot and construct its discrete state space. To obtain its discrete state space, a robot r, on one hand, needs a priori knowledge of its own path and can broadcast it to the robots within its communication range; on the other hand, r should communicate to all robots that are connected to it through a multi-hop communication path. Thus, with a sequence of communication, r can retrieve the collision segments and construct its discrete model. If r is not connected to a robot r' through a multi-hop path, it implies that r' is far away from r. Hence, the motion of r' cannot affect that of r and r need not consider the motion of r'. Once r' can communicate with r, r refines its discrete model with the new information. Thus, each



(a) Original path. (b) Decomposed path.

FIG. 5.7: Decomposition of a path with multiple circuits.

robot can build its model in a distributed way. It does not need any global information and is adaptive to the change in the number of robots.

In the following three chapters, we study robot motion control in terms of the discrete models built in this chapter. With discrete models, though we simplify the motion control of multi-robot systems, we guarantee that robots can always avoid unsafe motion, especially deadlocks, which is hard to avoid during the continuous motion planning. The major impact of path discretization is its restriction on the high-level transitions by ignoring the low-level maneuvers, which, however, can be complemented by the local continuous controllers. Indeed, for the high-level control, we always assume that suitable local continuous controllers are available that take into account robots' dynamics and can stop robots' motion in a short time, which can realize the high-level decisions. For example, once a robot moves into a state by firing a $move$ transition, its local continuous controller generates a proper velocity to move at the state.

Our discussion in this thesis focuses on paths satisfying that each discrete state is passed once in each round. For paths whose discrete states may be passed multiple times in a round, we can decompose it into an equivalent path containing only one circuit based on its unique motion. Suppose $s_j \in S^i$. Let $d^i(s_j)$ be the number of occurrences of s_j in the discrete path of r_i. Then the decomposition can be done by decomposing s_j into $d^i(s_j)$ sub-states, say $s_{j,1}, s_{j,2}, \ldots, s_{j,d^i(s)}$, each of which replaces the original s_j in the unique path of the sub-graph. To broadcast its state sequence to others, r_i still uses s_j; if it receives a state sequence containing s_j, r_i replaces this state with the nearest sub-state from its current state. Thus, such a decomposition cannot affect others. For example, consider the path shown in Fig. 5.7. Suppose r_i has a path $s_1 \to s_2 \to s_3 \to s_4 \to s_2 \to s_3 \to s_1$, as shown in Fig. 5.7(a). We decompose s_2 into s_{21} and s_{22}, and s_3 into s_{31} and s_{32}. The resulting path is $s_1 \to s_{21} \to s_{31} \to s_4 \to s_{22} \to s_{32} \to s_1$, shown in Fig. 5.7(b).

Chapter 6

Distributed Approach to Collision and Deadlock Avoidance in Multi-Robot Systems

Based on the LTS model obtained in Chapter 5, in this chapter, we focus on distributed collision and deadlock avoidance in a multi-robot system with fixed paths.

6.1 Introduction

Due to intersections among paths, robots may collide with others during their motion. Besides, during the procedure of collision avoidance, deadlocks may occur. For example, as shown in Fig. 6.1, there are four autonomous vehicles passing through an unsignalled traffic intersection. At the current time instant, the four vehicles are in a deadlock, and the traffic is completely jammed.

If robots can replan their paths during their motion, collisions and deadlocks can be resolved by changing robots' paths, such as the work in Chapter 4. However, when each robot in a system is required to move along a predefined path and cannot change its path or motion direction, the only way to avoid collisions and deadlocks is that different robots move to the same position at different times. Two common control structures

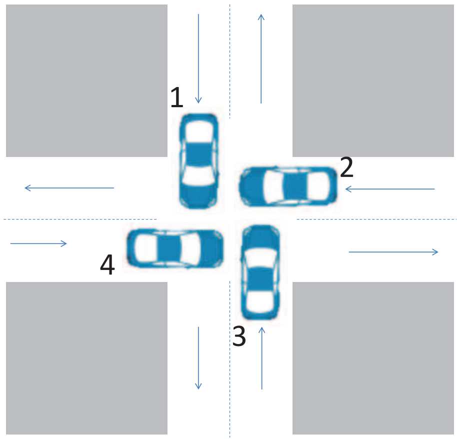


FIG. 6.1: A deadlock among 4 vehicles at an intersection.

for collision and deadlock avoidance are centralized control and decentralized control. However, centralized control lacks capability of flexibility and scalability, while decentralized control sometimes is conservative. Based on the LTS models from Chapter 5, in this chapter, we propose a distributed algorithm to avoid collisions and deadlocks. Under this algorithm, each robot repeatedly checks whether its current $move$ transition can be fired. First, a robot executes its own local mechanism to independently avoid collisions by checking whether its succeeding state is occupied. Despite its applicability to avoid collisions, such a scheme is so simple, if not naive, that deadlocks may occur. Second, the robot further checks whether the one-step move of a robot can cause a deadlock. If "yes", the algorithm will control the robot to fire its $stop$ transitions and the robot stops at its current state.

The main contribution of this work is a real-time and distributed algorithm to avoid collisions and physical deadlocks in multi-robot systems. It has the following advantages. First, robots can execute the algorithm in a distributed manner. Each robot only needs to communicate with its neighbors within two states to exchange their current states and verify collisions and deadlocks. Thus, it can avoid state explosion. Second, it has sound scalability and adaptability. This means that the algorithm can be adaptive to the change of robot quantity in the system. Thus, it is available to increase or decrease robots during the execution of the system. Third, this algorithm is maximally permissive for the motion of robots in terms of the high-level abstraction. Thus, each robot in the multi-robot system can achieve high performance in terms of high-level abstraction, i.e., they can stop as less as possible and move as smoothly as possible.

This chapter is organized as follows. The persistent motion problem of the system is also stated in this section. A distributed algorithm for collision avoidance is presented in Section 6.3. In Section 6.4, we propose an improved distributed algorithm for both collision and deadlock avoidance. The simulation results and implementation are described in Section 6.5. Section 6.6 gives some discussion about our work. Finally, conclusions and future work are discussed in Section 6.7.

6.2 Problem Statement

In this section, we state the problem studied in this chapter. We first give definitions of collisions and deadlocks in terms of LTS models and then give the problem statement.

As described in Theorem 2, if two robots are in a collision, then they are at the same state. So if two robots are not at the same state, they are not in a collision. Hence, we have the following collision definition in terms of discrete LTS models.

Definition 11 (Collision). A multi-robot system $\mathcal{T}$ is in a collision if there exist at least two robots r_i and r_j, $i \neq j$, such that $s^i_{cur} = s^j_{cur}$, where s^i_{cur} and s^j_{cur} are their current states, respectively.

Based on the description of deadlocks in [145], we have the following definition.

Definition 12 (Deadlock). A multi-robot system $\mathcal{T}$ is in a deadlock at configuration c if some robots, $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$, are in a circular wait: $\forall i_m \in \{i_1, \ldots, i_k\}$, $Pos_{i_m}(c(i_m)) = c(i_{m+1})$, where $i_{k+1} = i_1$.

Suppose the set of all configurations in $\mathcal{T}$ is $\mathcal{C}$, the set of collision configurations is $\mathcal{C}_c$, and the set of deadlock configurations is $\mathcal{C}_d$. Then, the set of safe configurations is $\mathcal{C}_s = \mathcal{C} \setminus (\mathcal{C}_c \cup \mathcal{C}_d)$. Using the logical operators "implication" ($\rightarrow$) and "conjunction" ($\wedge$), and temporal operators "eventually" ($\Diamond$) and "always" ($\Box$), we can give the problem statement.

Problem 2. Given the LTS models $\{\mathcal{T}_i\}_{i \in \mathbb{I}_N}$ of the robots in a system, find a distributed and real-time motion control policy for the system such that any reachable configuration c satisfies: $(c \in \mathcal{C}_s) \wedge (\wedge_{i \in \mathbb{I}_N} \Box(c(i) \rightarrow \Diamond \neg c(i)))$.

The first formula means all reachable configurations are safe, i.e., there are no collisions or deadlocks, and the second one means each robot can do persistent motion and cannot stay at the same state forever.

6.3 Collision avoidance

In this section, we propose a distributed algorithm to avoid collisions among robots. The main idea is that if it predicts a collision with another robot after the next $move$ transition, a robot stops itself to wait for the move of that one. Next, we give the detailed description.

Based on Definition 11, a system is collision-free if and only if $\forall s \in S_\alpha$, there exists at most one robot at s. Let a boolean signal $sign_s$ denote the status of s. If s is not occupied by any other robots, $sign_s = 0$; otherwise, $sign_s = 1$. A robot is movable to s if s is a private state or $sign_s = 0$.

However, since each robot checks its succeeding state independently, several robots can move to the same state simultaneously. Thus, to guarantee mutual exclusion, they should negotiate with each other to determine which one can actually move. There are different negotiation strategies. For example, a possible negotiation strategy is random selection, which can be implemented as follows. Suppose X, called *negotiation region*, is a set of successive states containing only collision states. At each time, robots can communicate with their neighbors to identify the robots that are moving into or in X. Let E_X be the set of movable robots that are moving in or into X. First, each robot in E_X generates a random time delay based on the same distribution. Then, they communicate with others to retrieve the delays, and the robot with the minimum one obtains the right to move. Once the robot is determined, E_X is reset to empty.

We use $NEG(E_X)$ to denote the negotiation process, which returns the robot that can move forward. Let $Sign = \{sign_s, s \in S_\alpha\}$ and $Sign(s) = sign_s$, and the collision avoidance strategy for r_i is as follows: when it is about to move to $s \in S_\alpha^i$, i.e., the current state is $Pre_i(s)$, r_i first checks the value of $Sign(s)$. If $Sign(s) = 0$, broadcast

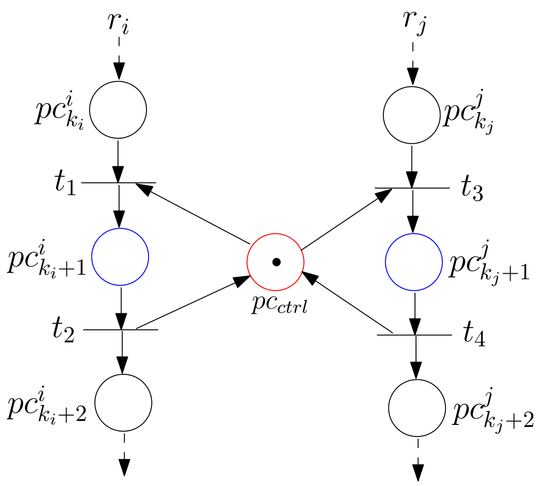


FIG. 6.2: Petri net description of collision avoidance between r_i and r_j.

its index to E_{X_s}, and execute $NEG(E_{X_s})$. If r_i gets the right to move, it enters s and the signal of s changes to 1. Otherwise, it stops at its current state.

We can describe this control framework in terms of Petri nets in a more intuitive way. As shown in Fig. 6.2, places $pc^i_{k_i} - pc^i_{k_i+2}$ (resp., $pc^j_{k_j} - pc^j_{k_j+2}$) represent three consecutive states of r_i (resp., r_j). Each transition represents the $move$ event from its input place to the output one. $pc^i_{k_i+1}$ and $pc^j_{k_j+1}$ represent the same state, say s, in $CS^{i,j}$. In order to avoid a collision, r_i and r_j cannot stay at $pc^i_{k_i+1}$ and $pc^j_{k_j+1}$ at the same time, i.e., for any reachable marking M, $M(pc^i_{k_i+1}) + M(pc^j_{k_j+1}) \leq 1$. We add a control place pc_{ctrl}, performing as the signal, i.e., $sign_s$. If $M(pc_{ctrl}) = 1$, $sign_s = 0$; otherwise, $sign_s = 1$. Only when pc_{ctrl} has a token may the transitions t_1 and t_3 be enabled. Indeed, when $M(pc^i_{k_i}) = M(pc^j_{k_j}) = M(pc_{ctrl}) = 1$, t_1 and t_3 are enabled simultaneously and can be fired. But only one of them can be fired. Thus, the firing selection performs the negotiation process, i.e., $NEG(E_X)$. With this comparison, the negotiation strategies among multiple robots can also be inspired by methods for the selection of firing transitions in Petri nets.

Based on the collision avoidance framework, the distributed algorithm to avoid collisions for robot r_i is shown in Algorithm 3. Note that in the algorithm, $Sign$ is a collection of local variables $sign_s$. Each robot stores its own local variables: $\{sign_s, s \in S^i_\alpha\}$. By checking its path, each robot can determine the values of $sign_s$ and execute the collision avoidance algorithm independently.

Algorithm 3: Collision avoidance algorithm for Robot r_i.

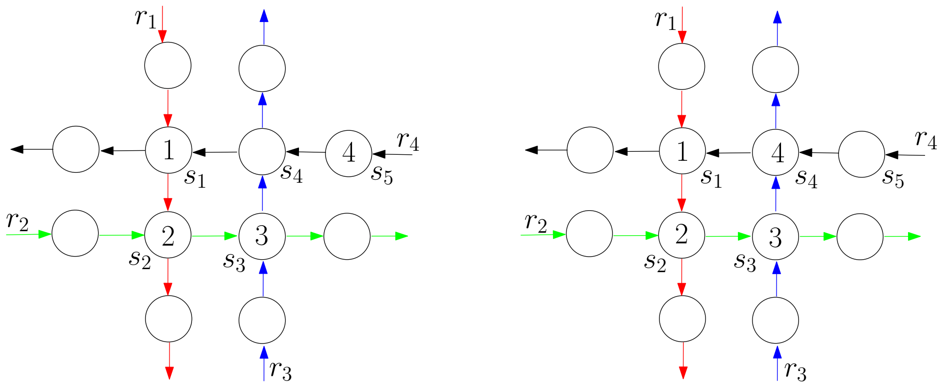
Input : $\mathcal{T}_i = \langle S^i, \Sigma_i, \rightarrow_i \rangle$, current state s^i_{cur}, and $Sign(s)$, $s \in S^i_\alpha$.
Output: No collision occurs during the motion of r_i.

Initialization: $s_{cur} = s^i_{cur}$, $s_{next} = Pos_i(s_{cur})$;
if $s_{next} \in S^i_\beta$ **then**
    Execute the transition $s_{cur} \xrightarrow{move}_i s_{next}$;
    if $s_{cur} \in S^i_\alpha$ **then**
        $Sign(s_{cur}) = 0$;
    $s_{cur} = s_{next}$; $s_{next} = Pos_i(s_{cur})$;
else if $Sign(s_{next}) == 0$ **then**
    Add r_i to E_{X_s};
    if $NEG(E_{X_s}) == r_i$ **then**
        Execute the transition $s_{cur} \xrightarrow{move}_i s_{next}$;
        if $s_{cur} \in S^i_\alpha$ **then**
            $Sign(s_{cur}) = 0$;
        $Sign(s_{next}) = 1$; $s_{cur} = s_{next}$; $s_{next} = Pos_i(s_{cur})$;
else if $Sign(s_{next}) == 1$ **then**
    Stop the motion at the current state;

6.4 Deadlock Avoidance

In Section 6.3, we have proposed a method to avoid collisions for each robot during its motion. Each robot checks independently whether its succeeding state is occupied. If "yes", it stops; otherwise, the robot moves to the succeeding state and prevents other robots from moving to this state. When multiple robots mutually prevent the moves of other robots, deadlocks may occur.

For example, consider the situation shown in Fig. 6.3. There are four robots r_1, r_2, r_3, and r_4. The states s_1, s_2, s_3, and s_4 are collision states between r_1 and r_4, r_1 and r_2, r_2 and r_3, and r_3 and r_4, respectively. Fig. 6.3(a) shows the current states of the four robots, i.e., $r_1 - r_4$ are at $s_1 - s_3$, and s_5, respectively. At the current moment, r_4 is near the end of the related segment and begins to execute its collision avoidance algorithm described in Algorithm 3. Since s_4 is empty, the signal $Sign(s_4) = 0$. Hence, the event $move$ in $\mathcal{T}_4$ is activated and causes r_4 to transit to s_4. The system reaches the configuration shown in Fig. 6.3(b). At this configuration, $r_1 - r_4$ are waiting for the



(a) Before the move of r_4.

(b) After the move of r_4.

FIG. 6.3: A situation that causes a deadlock among four robots.

move of r_2, r_3, r_4, and r_1, respectively. They are in a circular wait. Thus, the system is in a deadlock.

6.4.1 Deadlock Avoidance Algorithm

In this subsection, we introduce an improved algorithm for the system to avoid both collisions and deadlocks. We first study some properties of deadlocks of the multi-robot system in terms of graph theory. For the preliminary knowledge of graph theory, readers can refer to reference [204].

Definition 13 (Directed Graph). Let $\mathcal{T}_i = \langle S^i, \Sigma_i, \rightarrow_i, s_0^i \rangle$ be the LTS model of robot r_i, $i \in \mathbb{I}_N$. A directed graph of the multi-robot system is a two-tuple $G = \langle V, E \rangle$, where

- $V = \cup_{i=1}^{N} S^i$ is the finite set of vertices;
- $E = \cup_{i=1}^{N} \rightarrow_{i,move}$ is the finite set of edges;

Remark 5. (1) In a directed graph, a directed edge e from v_i to v_{i+1} is denoted as (v_i, v_{i+1}), and v_i is designated as the tail and v_{i+1} is designated as the head.

Based on Proposition 2, the undirected graph with the same topology structure of G is a simple graph. Thus, we have the following definitions.

Definition 14 (Cycle). Let $G = \langle V, E \rangle$ be the directed graph of a multi-robot system. A cycle of G is a sequence $\langle v_1, e_1, \ldots, v_n, e_n, v_1 \rangle$ such that (1) $\forall i \in \mathbb{I}_n = \{1, 2, \ldots, n\}$,

$v_i \in V$, and $e_i = (v_i, v_{i+1}) \in E$ is the directed edge from v_i to v_{i+1}, where $v_{n+1} = v_1$; (2) $\forall i_1, i_2 \in \mathbb{I}_n$, $v_{i_1} \neq v_{i_2}$ if $i_1 \neq i_2$; and (3) $\forall j_1, j_2 \in \mathbb{I}_n$, suppose $e_{j_1} \in \rightarrow_{k_1,move}$ and $e_{j_2} \in \rightarrow_{k_2,move}$, $k_1 \neq k_2$ if $j_1 \neq j_2$.

For example, as the system shown in Fig. 6.3, the sequence $\langle s_1, (s_1, s_2), s_2, (s_2, s_3), s_3, (s_3, s_4), s_4, (s_4, s_1), s_1\rangle$ is a cycle of the system. There are four different vertices representing four different states, i.e., s_1, s_2, s_3, and s_4, and four edges representing transitions of different robots, i.e., $(s_1, s_2) \in\rightarrow_{1,move}$, $(s_2, s_3) \in\rightarrow_{2,move}$, $(s_3, s_4) \in\rightarrow_{3,move}$, and $(s_4, s_1) \in\rightarrow_{4,move}$.

In the directed graph of a multi-robot system, a vertex can be occupied by different robots at different times. Since each robot has its unique motion direction, there may be no deadlock even if some robots are in a cycle. Consider the two configurations shown in Figs. 6.4(a) and 6.4(b). The robots at either configuration are in a cycle. But the robots in Fig. 6.4(b) are deadlock-free. In fact, only some cycles satisfying certain conditions can cause deadlocks. In the sequel, we first give the definition of deadlock cycles, and then prove that only deadlock cycles can cause deadlocks.

Definition 15 (Active Edge). Given the graph $\langle V, E\rangle$ of a multi-robot system, a directed edge e, $e = (s_1, s_2) \in \rightarrow_{i,move} \subset E$, is called an active edge if the robot r_i is at s_1.

Definition 16 (Deadlock Cycle). A deadlock cycle is a cycle where all edges are active edges.

For example, the four robots in Fig. 6.4(a) constitute a deadlock cycle since each robot is at the tail of the corresponding edge and thus each edge is an active edge. The robots in Fig. 6.4(b) do not constitute a deadlock cycle although each vertex of the cycle is occupied by a robot.

Theorem 3. A multi-robot system is in a deadlock if and only if some robots compose a deadlock cycle.

Proof. Sufficiency: A subset of robots, say $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$, construct a deadlock cycle in the corresponding graph. Based on Definitions 14 and 16, we suppose that the cycle is the sequence $\langle s_{r_{i_1}}, e_{i_1}, s_{r_{i_2}}, e_{i_2}, \ldots, s_{r_{i_k}}, e_{i_k}, s_{r_{i_1}}\rangle$, where the robot r_{i_j} is at $s_{r_{i_j}}$ and the edge $e_{i_j} = (s_{r_{i_j}}, s_{r_{i_{j+1}}})$ is an active edge, i.e., $e_{i_j} \in \rightarrow_{i_j,move}$. The cycle is shown in

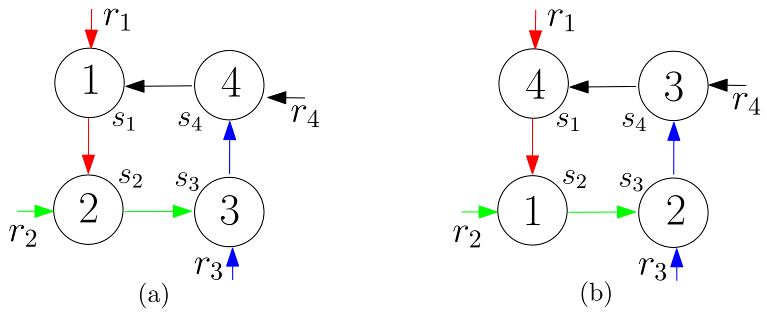


FIG. 6.4: Two kinds of cycles in the directed graph of a multi-robot system. (a) A deadlock cycle. (b) A cycle but not a deadlock cycle.

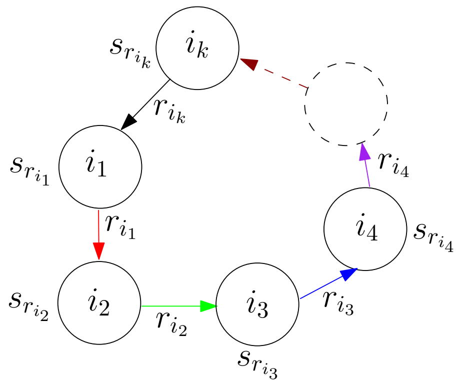


FIG. 6.5: k robots in a deadlock cycle.

Fig. 6.5. We can conclude that these k robots are in a circular wait and cannot move any more. Indeed, r_{i_1} cannot move since it can only move to state $s_{r_{i_2}}$, which is occupied by robot r_{i_2}. So r_{i_1} needs to wait for the move of r_{i_2}. At the same moment, since its succeeding state, i.e., $s_{r_{i_3}}$, is occupied by robot r_{i_3}, r_{i_2} cannot move until r_{i_3} moves away from r_{i_2}'s path. However, r_{i_3} also cannot move forward at the same time since r_{i_4} is at $s_{r_{i_4}}$, i.e., the succeeding state of r_{i_3}. By going forward until r_{i_k}, we find that the succeeding state of r_{i_k} is occupied by r_{i_1}, leading to the stoppage of r_{i_k} at the current state. Thus, all of them are in a circular, and the system is in a deadlock.

Necessity: To prove by contradiction, we hypothesize that the system is in a deadlock but with no deadlock cycles. However, in the case there is no deadlock cycle, we can prove that each robot can move one step forward eventually. Consider an arbitrary robot r_i. Suppose r_i is at s_{r_i}. If its succeeding state is empty, r_i can move forward. If the succeeding state is occupied by a robot, say r_{i_1}, let's consider r_{i_1}'s succeeding state. If this state is empty, r_{i_1} can move forward. After the move of r_{i_1}, r_i can move forward. Otherwise, suppose the state is occupied by a robot, say r_{i_2}. Clearly, we have

$i_2 \neq i_1$ and $i_2 \neq i$; otherwise, there is a deadlock cycle. We continue to consider r_{i_2}'s succeeding state and check whether it is occupied by any robot. If r_{i_2}'s succeeding state is empty, r_{i_2}, r_{i_1}, and r_i can move forward in sequence. Instead, if it is occupied by a robot, say r_{i_3}, we have $i_3 \neq i_2$, $i_3 \neq i_1$, and $i_3 \neq i$; otherwise, there is a deadlock cycle. We next need to check whether the succeeding state of r_{i_3} is occupied by a robot or not. Do the same analysis for the remaining robots one by one by repeating the previous procedures. Since the number of robots is finite, we can end with a robot whose succeeding state is empty; otherwise, it can compose a deadlock cycle among some robots. Thus, the robots can move forward in turns and at last r_i moves forward. By far, we can conclude that every robot can move forward. This is a contradiction to the precondition that the system is in a deadlock. Hence, there exists a deadlock cycle. □

From Theorem 3, we can resolve deadlocks by avoiding deadlock cycles. Next, we study how to avoid deadlock cycles and then give the collision and deadlock avoidance algorithm. Here we just consider the direct deadlocks, while in the future we will consider the impending deadlocks.

Before giving the algorithm, we describe the distributed procedure to detect deadlock cycles. Suppose r_i is at s_{r_i}. First, r_i checks its succeeding state $Pos_i(s_{r_i})$. If there exists r_{i_1} such that $s^{i_1}_{cur} = Pos_i(s_{r_i})$, a message (r_i, s_{r_i}, i_1) is delivered to r_{i_1}. r_{i_1} begins to estimate its succeeding state after receiving the message. If $Pos_{i_1}(s^{i_1}_{cur})$ is also occupied by a robot, say r_{i_2}, r_{i_2} can receive the corresponding message (r_i, s_{r_i}, i_2) and begin to estimate the succeeding state. Continue delivering the message until there exists a robot r_{i_k} whose succeeding state is either s_{r_i} or idle. The former means there is a deadlock cycle when r_i is at s_{r_i}, while the latter means r_i's $move$ transition to s_{r_i} cannot construct a deadlock cycle. The details are shown in Algorithm 4. In the algorithm, $f(s_{r_i}, \rightarrow_j)$ is a function of r_j to check whether its succeeding state is s_{r_i}. If the succeeding state is s_{r_i}, the check process is finished and r_i determines that there is no deadlock cycle. Otherwise, it returns (r_i, s_{r_i}, k) if its succeeding state is occupied by r_k; while returns $(r_i, s_{r_i}, 0)$ if its succeeding state is not occupied by any robots.

The validation of the algorithm is given through the following theorem.

Theorem 4. Algorithm 4 can always end by returning a boolean value at any time.

Algorithm 4: Deadlock cycle detection algorithm for r_i: $Detect(\mathcal{T}_i, s_{r_i})$.

Input : LTS model $\mathcal{T}_i$, the state needed to detect s_{r_i}.
Output: A boolean value. /* false: No deadlock cycle is detected if r_i is at s_{r_i}; true: r_i at s_{r_i} can cause a deadlock cycle. */

1 Initialization: $r_j = r_i$;
2 **while** *true* **do**
  /* r_j checks its succeeding state. */
3   $(r_i, s_{r_i}, k) = f(s_{r_i}, \rightarrow_j)$;
4   **if** $Pos_j(s^j_{cur}) == s_{r_i}$ **then**
5   **return** true;
6   **else if** $k == 0$ **then**
7   **return** false;
8   **else**
    /* r_j sends the message (r_i, s_{r_i}, r_k) to r_k. */
9     $j = k$;

Proof. From the proof of Theorem 3, for any robot r_j, there exists a robot such that its succeeding state either is free or is occupied by r_i after a finite number of message deliveries. Note that in the *while* loop of Algorithm 4, each loop is a message delivery. Thus, one of the conditions in Lines 4 and 6 of Algorithm 4 can eventually be satisfied after a finite number of loops. Since there are N robots in the system, the maximal number of loops is N. □

From Algorithm 4, we notice that every time each robot only needs to check its next two states to determine whether its move could cause a deadlock cycle. Hence, each robot only needs to communicate with the robots that are at its next two consecutive states. Thus, each robot only requires a communication range within two states.

Based on the definition of deadlock cycles, we can infer that the move of a robot may cause a deadlock cycle only when its next two consecutive states are both collision states. Thus, Algorithm 4 only needs to be executed when robot r_i is at a state s satisfying $Pos_i(s) \in S^i_\alpha$ and $Pos_i(Pos_i(s)) \in S^i_\alpha$. When it is at s, r_i needs to predict whether its move can cause a deadlock cycle before proceeding ahead. If a deadlock cycle is predicted, the robot cannot move forward. The detailed collision and deadlock avoidance algorithm is shown in Algorithm 5. Note that since each robot checks deadlock cycles in a distributed way, there may be many robots that can move forward at

Algorithm 5: Collision and deadlock avoidance algorithm for r_i.

Input : The LTS model $\mathcal{T}_i$, current state s_{cur}, and local signals $Sign(s)$, $s \in S^i_\alpha$.
Output: No collisions and deadlocks occur during the motion of r_i.

1 Initialization: $s_{next1} = Pos_i(s_{cur})$, $s_{next2} = Pos_i(s_{next1})$, set the negotiation region X;
2 **if** $s_{next1} \in S^i_\beta$ **then**
3   Execute the transition $s_{cur} \xrightarrow{move}_i s_{next1}$;
4   **if** $s_{cur} \in S^i_\alpha$ **then**
5         $Sign(s_{cur}) = 0$;
6     $s_{cur} = s_{next1}$; $s_{next1} = Pos_i(s_{cur})$; $s_{next2} = Pos_i(s_{next1})$;
7 **else if** $Sign(s_{next1}) == 0$ **then**
8   **if** $(s_{next2} \in S^i_\beta) \vee (Sign(s_{next2}) == 0)$ **then**
9   Add r_i to E_X ;
10   **else if** *!Detect*$(\mathcal{T}_i, s_{next1})$ **then**
11   Add i to E_X;
12   **else**
13   r_i cannot move forward;
14   **if** $NEG(E_X) == r_i$ **then**
15         $E_X = \emptyset$;
16   Execute the transition $s_{cur} \xrightarrow{move}_i s_{next1}$;
17   **if** $s_{cur} \in S^i_\alpha$ **then**
18             $Sign(s_{cur}) = 0$;
19         $s_{cur} = s_{next1}$; $s_{next1} = Pos_i(s_{cur})$; $s_{next2} = Pos_i(s_{next1})$;
20         $Sign(s_{cur}) = 1$;
21 **else**
22   r_i cannot move forward;

the same time. Thus, these robots should negotiate with others and only one can move forward because of concurrency.

Now, let's take the system in Fig. 6.3(a) as an example to explain the distributed execution of Algorithm 5 in a multi-robot system. First, $r_1 - r_4$ perform this algorithm simultaneously. r_1 and r_2 find that they have to stop at their current states since their succeeding states are occupied (Lines 21 and 22). r_3 finds that it is able to move forward based on Lines 8 and 9. Since s_1 is occupied, r_4 calls Algorithm 4 and sends the information (s_4, r_4) to r_1. Then, r_1 sends this information to r_2, and r_2 sends it to r_3. r_3 finds its succeeding state is s_4, and thus sends to r_4 the information that a deadlock is found. When r_4 received it, *Detect*$(\mathcal{T}_4, s_4) =$ true. So r_4 cannot be movable (Line 13).

Hence, $E_X = \{r_3\}$. Clearly, $NEG(E_X) = r_3$. So r_3 moves forward. Thus, with the deadlock avoidance algorithm, the situation shown in Fig. 6.3(b) cannot occur.

6.4.2 Performance Analysis of the Algorithm

Now we give the performance analysis of the proposed collision and deadlock avoidance algorithm, including the effectiveness and permissiveness analysis. For the sake of simplicity, we assume that the solution to resolve a deadlock cycle cannot cause any other deadlock cycles. This means if robot r_i finds that its move to s can cause a deadlock cycle with a set of robots, including the robot r_j satisfying $Pos_j(s^j_{cur}) = s$, then r_j can pass through s without causing deadlocks at some future moment. Thus, we have the following conclusions.

Theorem 5 (Effectiveness). Each robot can execute persistent motion without causing any collisions or deadlocks under the control of Algorithm 5.

Proof. Suppose r_i is at s. Lines 21 and 22 in Algorithm 5 guarantees that each reachable configuration based on Algorithm 5 is collision-free. Lines 12 and 13 in Algorithm 5 guarantees that the move of r_i cannot cause deadlock cycles. Thus, each reachable configuration based on Algorithm 5 is deadlock-free. Hence, the first requirement in Problem 2 is always satisfied. Now consider the second requirement. If r_i can eventually move one step forward, the proposition $s \to \lozenge\neg s$ is satisfied. The arbitrariness of s guarantees that $\square(s \to \lozenge\neg s)$ is satisfied for r_i. Applying this conclusion to all robots, we can conclude the second requirement is satisfied. Thus, we now only need to consider the situations that r_i cannot move forward at s. Indeed, there are two such situations in the algorithm: (1) $Detect(\mathcal{T}_i, Pos_i(s)) = 1$, and (2) there exists a robot r_{i_1} such that $Pos_i(s) = s^{i_1}_{cur}$. We need to prove that r_i can eventually move forward in either situation.

For the first case, there exist a set of robots r_{i_1}, r_{i_2}, …, r_{i_k} such that $s^{i_{j+1}}_{cur} = Pos_{i_j}(s^{i_j}_{cur})$, $j = 1, 2, \ldots, k-1$, and $Pos_{i_k}(s^{i_k}_{cur}) = Pos_i(s) \triangleq ss$ is empty. Based on the assumption declared at the begin of this subsection, r_{i_k} can move to ss and then to $Pos_{i_k}(ss)$ in the future. When r_{i_k} arrives at $Pos_{i_k}(ss)$, $Detect(\mathcal{T}_i, Pos_i(s)) = 0$

because $Pos_{i_{k-1}}(s_{cur}^{i_{k-1}})$ is now empty. Thus there is no deadlock cycle when r_i is at $Pos_i(s)$. Hence, r_i can move one step forward.

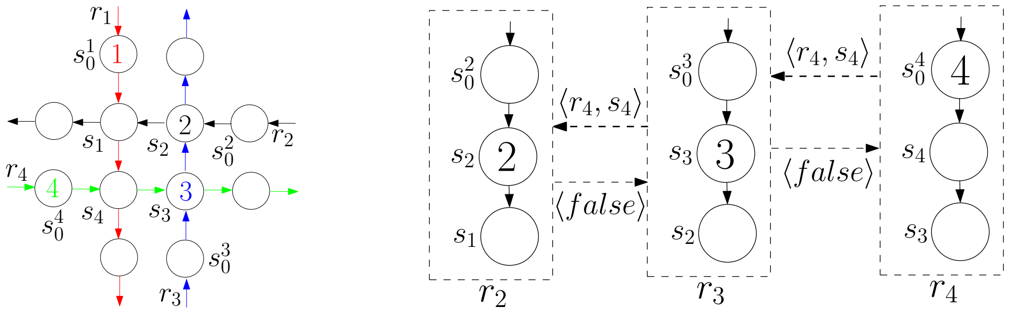
For the second case, there exist robots r_{i_1}, r_{i_2}, …, r_{i_k} satisfying $Pos_{i_1}(s_{cur}) = s^{\bullet i}$ and $s_{cur}^{i_{j+1}} = Pos_{i_j}(s_{cur}^{i_j})$ for $j = 1, 2, \ldots, k-1$. Moreover, $Pos_{i_k}(s_{cur}^{i_k})$ is empty. Otherwise there must exist a deadlock cycle, which should be detected and resolved in advance. Thus, r_{i_k} either can move forward or is in the first situation. As described before, r_{i_k} can finally move forward. After r_{i_k} moves forward, $r_{i_{k-1}}$ is in the same situation as r_k was. Thus, $r_{i_{k-1}}$ can move forward as a consequence. One by one, and finally r_i can move forward. □

Definition 17 (Admissible Motion). For any robot r_i with the LTS model $\mathcal{T}_i$, an admissible motion is the firing of a $move$ transition that cannot cause any collision and deadlock.

Theorem 6 (Maximal Permissiveness). The control policy described by Algorithm 5 is a maximally permissive control policy for r_i's motion.

Proof. Because of the concurrency, the admissible motion is described in terms of reachability. This means even though its current motion is admissible, the robot actually cannot move forward at some rounds since it does not win in the negotiation processes. During the computation of reachable graph, we need to list all the possible moves of the robots in E_X. Thus, considering the motion of r_i, we assume that r_i always wins the negotiation during our proof.

We need to prove that any possible control policies must contain the stopping motion of Algorithm 5. Suppose r_i is at an arbitrary state s at the current moment. On one hand, from the algorithm, r_i will stop its motion in two cases: (1) $Detect(\mathcal{T}_i, Pos_i(s)) = 1$ (Lines 12 and 13), and (2) $Pos_i(s) \in S_\alpha^i \wedge Sign(Pos_i(s)) = 1$ (Lines 21 and 22). The first one means that r_i's move can cause a deadlock cycle. Based on Theorem 3, such a move can lead the system to a deadlock. The second means r_i's current succeeding state is occupied by a robot. Thus, it cannot move forward in order to avoid collisions. Clearly, these two kinds of motion must be forbidden. This means that any available control policies for r_i must contain these two situations of stopping motion. On the other hand, except such two cases, r_i can always move forward based on the previous



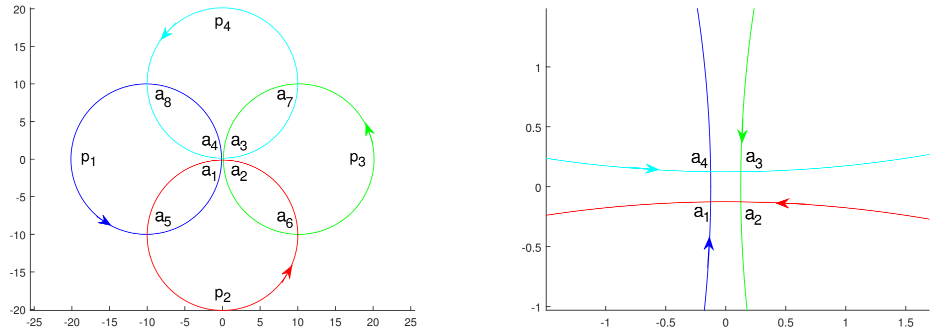
(a) Current states of the four robots. (b) Communication activated by r_4 for deadlock detection.

FIG. 6.6: An example to show communications among robots for deadlock detection.

assumption. Thus, for any state s, if r_i stops at s under Algorithm 5, r_i stops at s under any other available control policies. Hence, the proposed algorithm is maximally permissive. $\square$

The motion of the system under a maximally permissive control is the maximally permissive motion. Here the maximally permissive motion is with respect to evolution of the LTS models. Moreover, as described in the proof of Algorithm 6, the maximal permissive motion means the reachable configuration space is maximal, but does not mean that a robot in the admissible motion can always move forward. Indeed, because of concurrency, even though it can be able to move forward, a robot may be still at its current state. This happens because the robot does not get the right to move forward in the negotiation process. But when computing the reachable space, though it is unnecessary, each time we need to list all possibilities that one movable robot moves forward while others stay at their current states, without considering the negotiation process.

To the end, we illustrate the communication among robots in order to detect deadlocks. Consider the situation in which four robots are passing through the crossing. The current states of these robots are shown in Fig. 6.6(a). Consider the execution of r_4. At this moment, after checking the status of s_3 and s_4, r_4 needs to determine whether its move to s_4 can cause deadlocks since s_3 is occupied. The communication via message delivery is shown in Fig. 6.6(b). First, r_4 sends the message $\langle r_4, s_4\rangle$ to r_3, and then r_3 sends it to r_2. When it receives this message, r_2 sends a Boolean value $false$ to r_4 since r_2's succeeding state is not occupied by any robot. Thus, r_4 can conclude that there are no deadlocks when it is at s_4, i.e., $detect(r_4, s_4) = false$.



(a) Four intersecting paths $p_1 - p_4$. (b) The enclosed region defined by $a_1 - a_4$.

FIG. 6.7: Paths of four robots in our simulation.

6.5 Simulation Implementation and Results

6.5.1 Simulation Case and Results

In this section, we implement the algorithms in MATLAB. Simulations are carried out for a multi-robot system with four robots r_1, r_2, r_3, and r_4, whose paths are shown in Fig. 6.7. Each path is a circle with a radius of 10 units. Their detailed equations are $p_1 : (x + a)^2 + y^2 = 10^2$ (the blue one), $p_2 : x^2 + (y + a)^2 = 10^2$ (the red one), $p_3 : (x - a)^2 + y^2 = 10^2$ (the green one), and $p_4 : x^2 + (y - a)^2 = 10^2$ (the cyan one), where $a = \sqrt{10^2 - (\frac{\pi}{25})^2} + \frac{\pi}{25}$. There are totally 8 intersection points, i.e., $a_1 - a_8$.

Based on Definition 1, the parametric equations of the four paths are $p_1 = p_1(\theta_1) = (-a + 10\cos 2\pi\theta_1, 10\sin 2\pi\theta_1)$, $p_2 = p_2(\theta_2) = (10\sin 2\pi\theta_2, -a + 10\cos 2\pi\theta_2)$, $p_3 = p_3(\theta_3) = (a + 10\cos 2\pi\theta_3, 10\sin 2\pi\theta_3)$, and $p_4 = p_4(\theta_4) = (10\sin 2\pi\theta_4, a + 10\cos 2\pi\theta_4)$, where $\theta_1, \theta_2, \theta_3, \theta_4 \in [0, 1]$. Each path is discretized using the discrete points shown in Table 6.1, where $N^* = \{0, 1, 2, \ldots, 249\}$, and robots move among these discrete points. Note that the footprint of a robot at (x_0, y_0) is $(x - x_0)^2 + (y - y_0)^2 = (\frac{\pi}{25})^2$ by considering the safe radius.

The values of the parameter of the 8 points on different paths are shown in Table 6.2. For example, consider point a_1. a_1 is an intersection point of p_1 and p_2. The parameter value of a_1 on p_1 is $499/500$, while on p_2, the value is $126/500$.

TABLE 6.1: Discrete Points of the Four Paths

Path	Values of Parameters of the Discrete Points
p_1	$\theta_1 = \frac{(2k+1)}{500}$, $k = (N^*\backslash\{61, 62, 187, 188\}) \cup \{61.5, 187.5\}$
p_2	$\theta_2 = \frac{2k}{500}$, $k = (N^*\backslash\{0, 1, 124, 125\}) \cup \{0.5, 124.5\}$
p_3	$\theta_3 = \frac{(2k+1)}{500}$, $k = (N^*\backslash\{62, 63, 186, 187\}) \cup \{62.5, 186.5\}$
p_4	$\theta_4 = \frac{2k}{500}$, $k = (N^*\backslash\{0, 125, 126, 249\}) \cup \{125.5, 249.5\}$

TABLE 6.2: Parameter Values of Collision Points on Each Path

Point	Values of Different Parameters			
	θ_1	θ_2	θ_3	θ_4
a_1	$499/500$	$126/500$	–	–
a_2	–	$124/500$	$251/500$	–
a_3	–	–	$249/500$	$376/500$
a_4	$1/500$	–	–	$374/500$
a_5	$376/500$	$249/500$	–	–
a_6	–	$1/500$	$374/500$	–
a_7	–	–	$126/500$	$499/500$
a_8	$124/500$	–	–	$251/500$

$\star$: "–" means the point is not on the path of the PCS.

We first simulate the motion of the system under the control of Algorithm 3. Consider two different initial configurations of the system. Case 1: the initial positions of $r_1 - r_4$ are $\theta_1 = 479/500$, $\theta_2 = 116/500$, $\theta_3 = 229/500$, and $\theta_4 = 356/500$, respectively. Case 2: the initial positions of $r_1 - r_4$ are $\theta_1 = 479/500$, $\theta_2 = 104/500$, $\theta_3 = 229/500$, and $\theta_4 = 354/500$, respectively.

In our simulation, the motion of each robot is implemented by the *timer* object in MATLAB. Thus, all robots can be executed concurrently.

From the simulation results, we find that robots with the initial states of Case 1 can

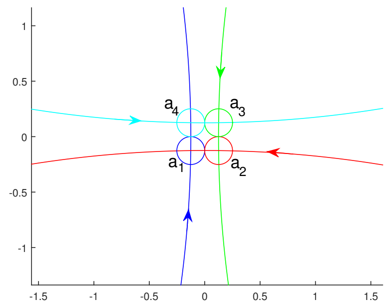
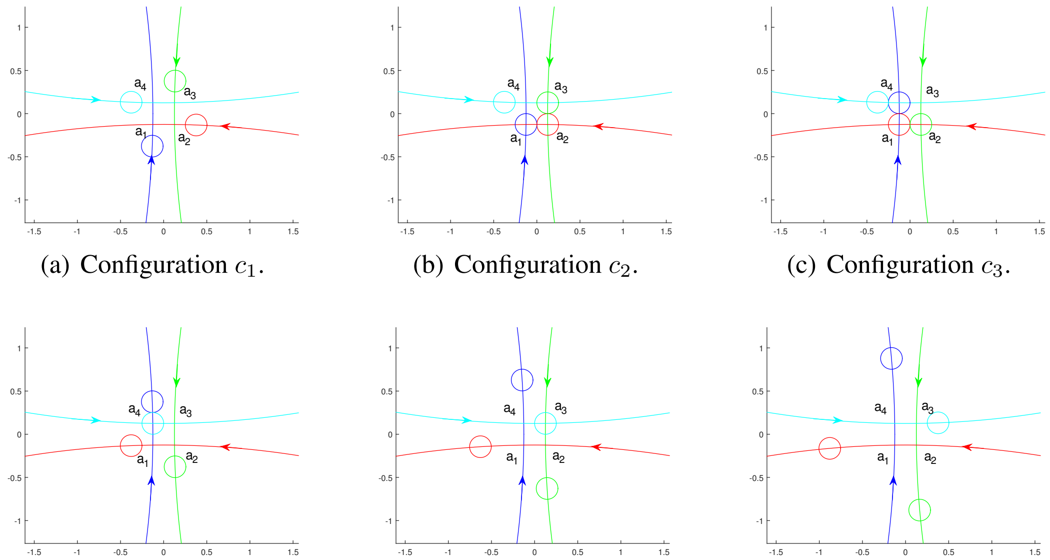


FIG. 6.8: A deadlock occurs in Case 2 under the control of the collision avoidance algorithm.



(a) Configuration c_1. (b) Configuration c_2. (c) Configuration c_3.

(d) Configuration c_4. (e) Configuration c_5. (f) Configuration c_6.

FIG. 6.9: Six snapshots of the simulation of Case 2 under control of deadlock avoidance algorithm. Configurations $c_2 - c_6$ show the process of deadlock avoidance.

move persistently without causing collisions and deadlocks; while with those of Case 2, after firing 10 transitions simultaneously, they stop at the configuration shown in Fig. 6.8. Clearly, at this configuration, a deadlock occurs. Thus, only the collision avoidance is not sufficient to guarantee the persistent motion of the system.

Next, we repeat the simulation of Case 2 by replacing Algorithm 3 with Algorithm 5. With this algorithm, the four robots need to negotiate with each other when they want to move to $a_1 - a_4$ simultaneously. Fig. 6.9 shows 6 snapshots of the simulation.

Suppose the system is now at the configuration shown in Fig. 6.9(a). At this moment, $r_1 - r_4$ are able to move one step forward based on the condition in Line 8 of Algorithm 5. Suppose r_1 wins in the negotiation process, r_1 moves one step forward

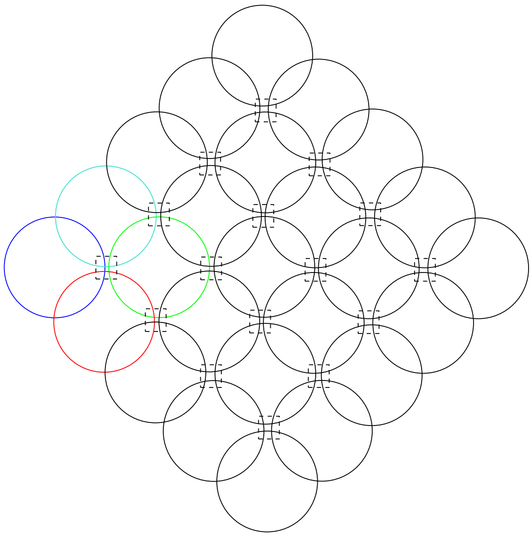


FIG. 6.10: Deadlocks in extended systems from 4 robots to 25 robots. There exist 16 deadlocks in the system. Each deadlock region is marked by a dashed square.

and reaches a_1. Then, $r_2 - r_4$ and r_1 are able to move forward. If r_2 is selected from their negotiation, it moves forward and arrives at a_2. Continually, r_3, r_4, and r_1 are able to move, but only r_3 is selected to move. Thus, r_3 arrives at a_3. Therefore, the system reaches the configuration shown in Fig. 6.9(b). At this configuration, r_4 predicts that its move to a_4 can cause a deadlock. Hence, r_4 cannot move based on the condition in Line 12 of Algorithm 5. Moreover, r_2 and r_3 cannot move forward based on Line 21 of their own local Algorithm 5. Thus, only r_1 can move one step forward based on Line 8 of its Algorithm 5. When r_1 reaches a_4, a_1 is empty. So r_2 is able to move forward and then is selected to move. The move of r_2 releases a_2 such that r_3 is allowed and selected to move to a_2. Thus, the configuration of the system is now shown in Fig. 6.9(c). At configuration c_3, r_4 cannot move forward since a_4 now is occupied by r_1. Since its next state is a private state, r_1 moves one step forward and leaves away from a_4, so do r_2 and r_3. Now r_4 can move one step forward since its next two consecutive states are empty. Suppose r_4 is selected to move one step forward, the system reaches the configuration shown in Fig. 6.9(d). We can do the similar analysis on how the system reaches the states shown in Figs. 6.9(e) and 6.9(f). When the system is at configuration c_6, we can conclude that it is effective to avoid collisions and deadlocks since all robots are at their own private states. The video for the simulation of Case 2 can be found at https://www.youtube.com/watch?v=fjosKjMXsW8.

TABLE 6.3: The Numbers of Robots and Different Deadlocks That May Occur

# robots	4	9	16	25	...	n^2	...
# deadlocks	1	4	9	16	...	$(n-1)^2$	...

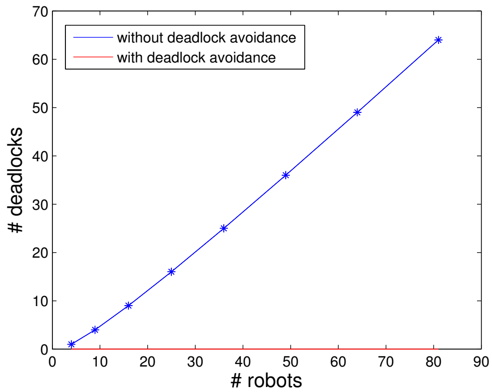
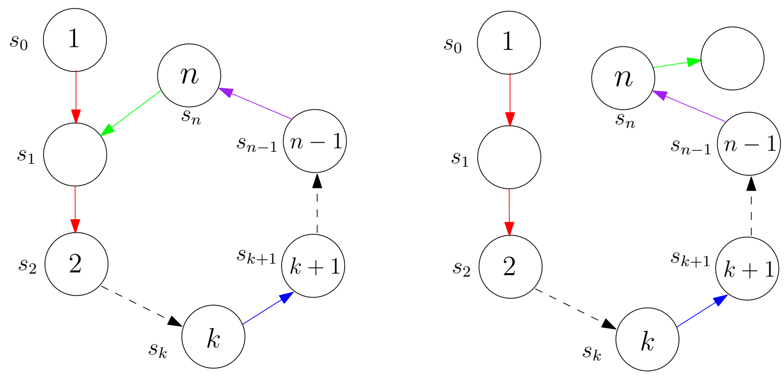


FIG. 6.11: The numbers of deadlocks that may occur in systems with different robots. Without deadlock avoidance, the number is linearly increased in proportion to the number of robots, while with our deadlock avoidance algorithm, there are no deadlocks during the evolution of the system.

For a deeper exploration of our algorithm, we first study deadlocks in the systems extended from the original system in Fig. 6.7(a) by continually adding the deadlock regions $p_1 - p_4$. For the first study, in an arbitrary extension, each path can intersect with at most four other paths, and each internal circle intersects with four other paths. A deadlock can only happen among four robots. Moreover, the paths of n^2 robots construct a square with n circles in each edge. For example, Fig. 6.10 shows an extended system with 25 robots. There are 16 deadlocks that may occur during the evolution of this system. The relation of the number of robots and that of deadlocks that may occur is shown in Table 6.3. We can find the number of deadlocks increases in proportion to the number of robots. Thus, the system would be at a great risk of breakdown if there are many robots in the system. With the control of proposed deadlock avoidance algorithm, there are no deadlocks that can occur during the evolution of the system, shown in Fig. 6.11. Hence, it is important to control a multi-robot system with the proposed deadlock avoidance algorithm, which is effective to avoid deadlocks.

Next, we would like to study the time for a robot to perform its deadlock detection process with different numbers of robots. As shown in Fig. 6.12, we study two configurations: the first one is that r_1 will detect a deadlock among n robots, and the second is that r_1 does not detect any deadlock among these robots since r_n's next state is not



(a) A configuraiton that a deadlock is detected by r_1 with n robots.

(b) A configuraiton that no deadlocks are detected by r_1.

FIG. 6.12: Two simulation configurations of n robots. (a) r_1 detects a deadlock with the other $n-1$ robots during its detection process. (b) r_1 does not detect a deadlock after a sequence of communications among the other $n-1$ robots.

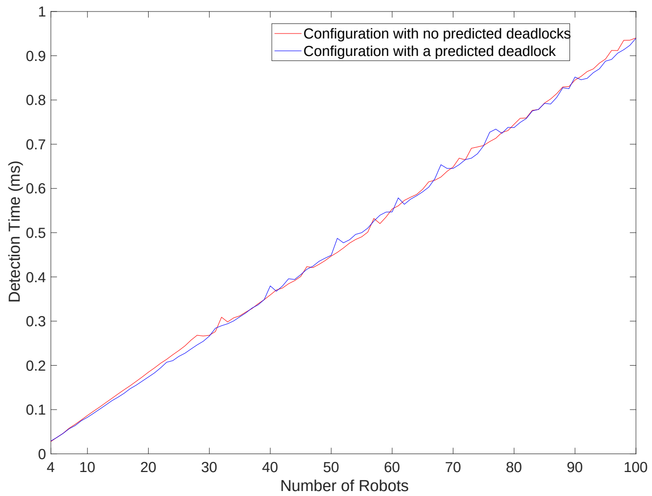


FIG. 6.13: Average computation time for either configuration with different numbers of robots.

s_1. We study different values of n, from 4 to 100. For each number, we run either configuration with 100 times and compute the average time. All our simulations are implemented with MATLAB R2017a on a desktop running Windows 10, and equipped with an Intel(R) Xeon(R) CPU E5-1650 v3 3.5GHz and 16 GB of RAM. The results are shown in Fig. 6.13. From the results, we can find that the computation time is almost increased linearly with respect to the number of robots. Indeed, based on our method, a robot involving a prediction process only needs to check the status of its next state and then transmits the message to another robot. If all robots are with the same configuration, each robot almost has the same time to perform its execution during the prediction process. Hence, the total computation time for a robot's prediction process increases linearly with respect to the number of robots involved.

(a) A real intersection. (b) Abstract model of the intersection.

FIG. 6.14: An intersection in NTU campus and its diagrammatic drawing.

6.5.2 Simulation Results on of a Practical Scenario

Now we simulate our algorithm on a scenario that four autonomous vehicles are passing through an intersection, such as the one shown in Fig. 6.14, which is an intersection in our campus.

Suppose four vehicles arrive at the intersection successively, as shown in Fig. 6.15. Fig. 6.16 shows the deadlock occurring among the vehicles only with the collision avoidance algorithm. Now we consider the evolution of the system with different deadlock avoidance algorithms. Fig. 6.17 shows an intermediate configuration of the system with the collision and deadlock avoidance algorithm proposed in [130]. Based on their method, the intersection is abstracted to one state, and at any time instant, there is at most one vehicle in the crossing. Thus, at the current time, even though they are able to move forward, vehicles 3 and 4 cannot move into the crossing since vehicle 2 is in the crossing. Fig. 6.18 shows three snapshots of the system during the move to pass through the crossing under the control of our method. From the configurations, we can find that vehicles 2, 3, and 4 can be in the crossing at the same time. At configuration 1 in Fig. 6.18(a), vehicle 1 cannot move in order to avoid deadlocks, while at configuration 2 in Fig. 6.18(b), vehicle 1 cannot move since it is stopped by vehicle 2. Only when vehicle 2 moves away can vehicle D move forward, shown in Fig. 6.18(c).

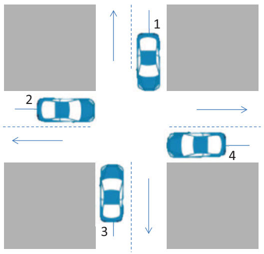


FIG. 6.15: Four vehicles arrive at the intersection successively.

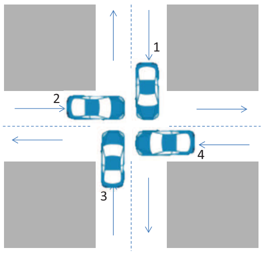


FIG. 6.16: Four vehicles are in a deadlock.

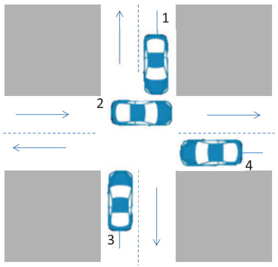
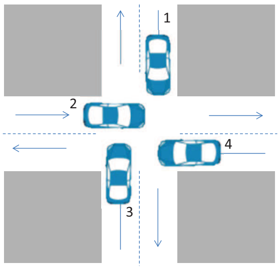
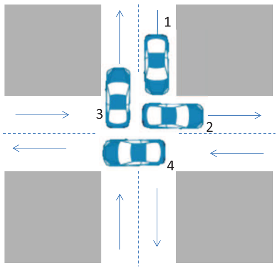


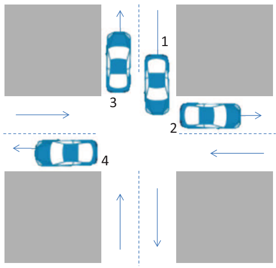
FIG. 6.17: An intermediate configuration under the method of [130].



(a) Configuration 1.



(b) Configuration 2.



(c) Configuration 3.

FIG. 6.18: Three snapshots of the motion under the control of our proposed algorithm.

6.6 Discussion

The most related work of this work is [130]. Authors in [130] divide all collision regions into a set of disjoint collision zones. Hence, each robot has at least one collision-free zone between any two collision zones. Collision avoidance is to control robots to enter the same collision zone at different times and collision avoidance does not cause any deadlocks. Then, they propose some centralized stop policies to determine the robots that need to stop entering the zone. Since different robots determine the sequence independently, there may exist decision making deadlocks. But there are no deadlocks physically. Thus, deadlocks can be resolved easily by resuming one of the robots. Due to the abstraction of disjoint collision zones, some admissible motion is forbidden.

For example, as shown in Fig. 6.19, there are four robots $r_1 - r_4$ to pass through a narrow and dense region. Taking the safe radius into consideration, r_1 can collide with $r_2 - r_4$ in the left, middle, and right segments, respectively; while $r_2 - r_4$ cannot collide with each other in this region. Based on the method in [130], this region is abstracted as one collision zone CZ^1, shown in Fig. 6.19(b). When it is in the segment $\widehat{A_1A_4}$,

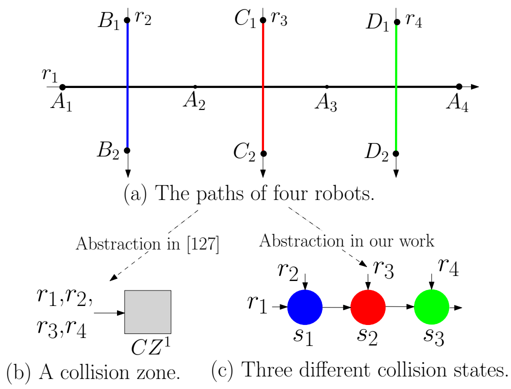


FIG. 6.19: Comparison of different ways to deal with collision regions in [130] and our work.

r_1 is in CZ^1; when it is in the segment $\widehat{B_1B_2}$, r_2 is in CZ^1; when it is in the segment $\widehat{C_1C_2}$, r_3 is in CZ^1; and when it is in the segment $\widehat{D_1D_2}$, r_4 is in CZ^1. Consider the following situation. Suppose $r_2 - r_4$ and r_1 arrive at B_1, C_1, D_1, and A_1 consecutively. r_2 enters into $\widehat{B_1B_2}$ first. When it moves into $\widehat{B_1B_2}$, r_2 is in CZ^1. So r_1, r_3, and r_4 have to stop their motion. Once r_2 leaves B_2, r_3 moves into $\widehat{C_1C_2}$, while r_1 and r_4 remain at a standstill. Next, when r_3 leaves C_2, r_4 moves into $\widehat{D_1D_2}$, but r_1 is still in halting. Only when r_4 is away from D_2 can r_1 start to move. Hence, r_1, r_3, and r_4 need more times of stop.

While with our method, this region is abstracted as three different states $s_1 - s_3$, shown in Fig. 6.19(c). When it is in the segments $\widehat{A_1A_2}$, $\widehat{A_2A_3}$, and $\widehat{A_3A_4}$, r_1 is at states s_1, s_2, and s_3, respectively; when it is in the segment $\widehat{B_1B_2}$, r_2 is at s_1; when it is in the segment $\widehat{C_1C_2}$, r_3 is at s_2; and when it is in the segment $\widehat{D_1D_2}$, r_4 is at s_3. Still, consider the former situation. When it moves into $\widehat{B_1B_2}$, r_2 arrives at s_1. So r_1 needs to stop to wait for the leaving of r_2. However, r_3 and r_4 can continue their motion since there are no robots at s_2 and s_3. So they do not need any stops. After r_2 leaves s_1, r_1 can move to s_1. Suppose the time for a robot to stay at a state is same. Thus, when r_1 is going to move to s_2, r_3 has left s_2. So r_1 can move to s_2 without any stops, and so does it for s_3. In conclusion, with our method, the four robots can pass through this region only with r_1's one time of stop. Hence, our method can lead to fewer stops from fewer robots.

At last, take the scenario given in Section 6.5.1 as an example to show the efficiency of our method and that in [130] in terms of the length of event sequences. We study 6 different initial configurations and count the length of the maximal event sequence

TABLE 6.4: Comparison of the Length of the Maximal Event Sequence Leading a Robot to Move 2 Rounds

Initial Configuration $(\times \frac{1}{500})$	Length of the Maximal Event Sequence		
	Optimal	Soltero's [130]	Ours
$(479, 104, 221, 348)$	496	499	498
$(471, 100, 229, 352)$	496	501	499
$(211, 456, 397, 478)$	496	496	496
$(327, \ 16, \ 77, 466)$	496	498	496
$(339, 378, 371, 196)$	496	496	496
$(479, 104, 229, 354)$	496	502	498

which leads a robot to move 2 cycles along its path. The results are shown in Table 6.4. Since the numbers of $move$ events of the four robots are the same, the shorter length of an event sequence, the fewer $stop$ events and the better concurrency and efficiency of the system. From Table 6.4, our method is an improvement of that in [130].

In conclusion, the method in [130] simplifies motion control of robots but it is conservative; while our method allows more admissible motion.

6.7 Conclusions

In this chapter, we investigate a real-time policy for collision and deadlock avoidance in a multi-robot system, where each robot has a predetermined and intersecting path. A distributed algorithm is proposed to avoid collisions and deadlocks in such a system. It is performed by repeatedly stopping and resuming robots whose next move can cause collisions or deadlocks. In the algorithm, each robot should check its next two consecutive states to determine whether it can move forward. We also prove that the proposed algorithm is maximally permissive for each robot's motion. The simulation results of a system with four robots further verify the effectiveness of the algorithm.

Chapter 7

Distributed Approach to Higher-Order Deadlock Avoidance in Multi-Robot Systems

In chapter 6, we focus on collision and deadlock avoidance in multi-robot systems where each robot has its own predetermined and closed path, by assuming that for some simple paths, deadlocks can be predicted and resolved directly. However, in some complex path networks, to avoid a deadlock may cause another circular wait, which results in higher-order deadlocks. A higher-order deadlock is a deadlock-free configuration, from which the system will lead to a deadlock inevitably. In this chapter, we investigate the characteristics of higher-order deadlocks and propose a distributed approach to avoiding high-order deadlocks.

7.1 Introduction

Deadlock avoidance is a crucial problem in motion control of multi-robot systems since deadlocks can crash the systems and degrade performance. Besides, sometimes, especially in the systems with fixed paths, deadlocks are not easy to be predicted since even though the system is deadlock-free at the current moment, it will fall into a deadlock in the future. Traditional methods to avoid such situations are either centralized, e.g., reachability graph based methods, or decentralized, e.g., the banker's algorithm or its variants. Centralized methods are effective to avoid all deadlocks but lack robustness and flexibility, while decentralized methods are efficient to avoid deadlocks but may forbid some available motion.

In this chapter, we investigate the structural properties of the configurations that will cause deadlocks inevitably by introducing the concepts of higher-order deadlocks and their orders, and propose a distributed approach to avoiding higher-order deadlocks. First, based on the LTS models built in Chapter 5, we conclude that there exist at most the $(N-3)$-th higher-order deadlocks with N robots. This means that deadlocks, if any, will occur unavoidably within $N-3$ steps of corresponding transitions. Second, a distributed algorithm is proposed to avoid higher-order deadlocks in the systems under our consideration. In the algorithm, each robot only needs to look ahead at most $N-1$ states, i.e., $N-3$ intermediate states and two endpoint states, to determine whether its move can cause higher-order deadlocks. To execute its local algorithm, a robot needs to communicate with its neighbors.

The main contributions of this work are twofold.

- The first one is that we propose the concept of deadlock orders. The main result is that N robots can form a higher-order deadlock with at most $(N-3)$-th order. This means from such a configuration, a deadlock will occur inevitably within $(N-3)$-step moves of the robots that are always included in a circuit.

- The second one is a distributed algorithm to avoid higher-order deadlocks. The algorithm allows only robots, whose one-step move cannot cause collisions and higher-order deadlocks, to move forward. According to the properties of higher-order deadlocks, each robot needs to look ahead at most $N-1$ states, i.e., one starting state, one ending state, and $N-3$ intermediate states, to determine whether it can move forward or not, rather than checks its whole state space.

This chapter is organized as follows. Section 7.2 gives the problem statement of deadlock avoidance in terms of LTSs. Sections 7.3 and 7.4 give a detailed control algorithm for higher-order deadlock avoidance and its distributed nature. Section 7.5

shows simulation results. Sections 7.6 and 7.7 give a detailed comparison with other typical methods and the conclusion, respectively.

7.2 Problem Statement

Before giving our problem statement, we recall some notations and definitions described in Chapter 6. Given a multi-robot system with N robots, whose LTS models are $\mathcal{T}_i, i \in \mathbb{I}_N$, its configuration, denoted as c, is a vector composing of the states of all robots in the system, i.e., $c = (s^1, s^2, \ldots, s^N)$, where $s^i \in S^i$ and $c(i) = s^i$. Recall the definitions of collision and deadlock configurations given in Definitions 11 and 12 in Section 6.2.

Definition 18. A configuration c is a collision one if $\exists i, j \in \mathbb{I}_N$, $i \neq j$, such that $c(i) = c(j)$. The set of collision configurations is denoted as $\mathcal{C}_c$, so the set of collision-free configurations is $\mathcal{C}_{cfree} = \mathcal{C} \setminus \mathcal{C}_c$.

Definition 19. A configuration c is a deadlock configuration if there exist a set of robots, $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$, such that $\forall i_m \in \{i_1, \ldots, i_k\}$, $c(i_m)^{\bullet i_m} = c(i_{m+1})$, where $i_{k+1} = i_1$. The set of deadlock configurations is denoted as $\mathcal{C}_d$, and the set of deadlock-free configurations is $\mathcal{C}_{dfree} = \mathcal{C}_{cfree} \setminus \mathcal{C}_d$.

A collision configuration is a configuration where there are at least two robots at the same state, and a deadlock configuration is configuration where some robots are in a circular wait. In the sequel, we can define higher-order deadlocks, the concentration of this chapter.

Definition 20 (Higher-order Deadlock). A configuration c is a higher-order deadlock if $(c \in \mathcal{C}_{dfree}) \wedge (c \to \Diamond c_d)$, where $c_d \in \mathcal{C}_d$. The set of higher-order deadlock configurations is denoted as $\mathcal{C}_{hdead}$.

For example, Fig. 7.1 shows an example of higher-order deadlocks. In Fig. 7.1, c_0 is a higher-order since the system will reach a deadlock eventually. Indeed, at c_0, r_1, r_3, and r_4 can move one step forward, leading to c_1, c_2, and c_3, respectively. At c_3, r_1, r_4, and r_5 form a deadlock. At c_1, a deadlock must occur among $r_1 - r_4$. If r_1 moves one step forward, $r_1 - r_3$ form a deadlock (c_4), while if r_3 moves forward, r_1, r_3, and r_4

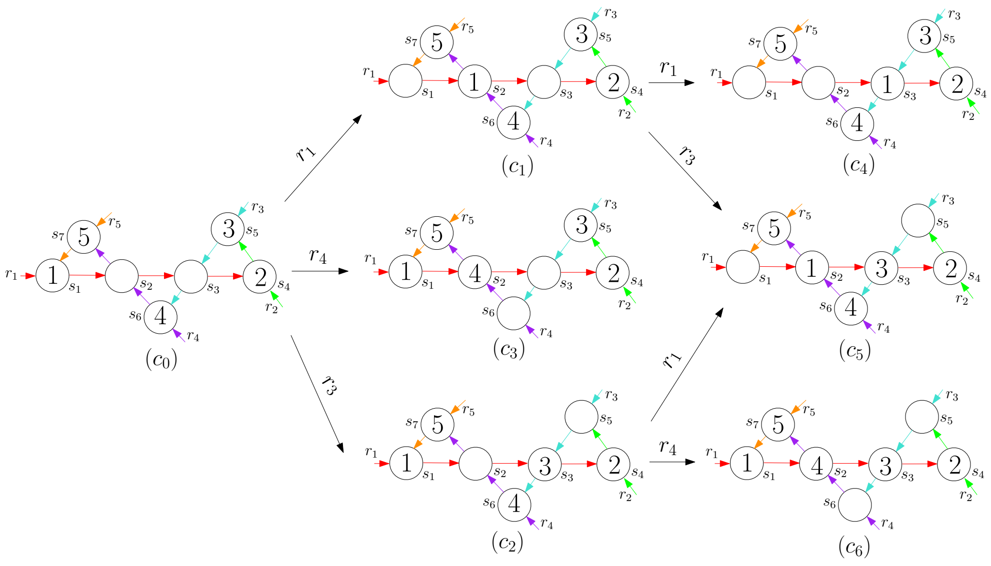


FIG. 7.1: An example of higher-order deadlocks. c_0 is a higher order deadlock. The system will finally lead to a deadlock, i.e., c_3, c_4, c_5, or c_6.

form a deadlock (c_5). At c_2, a deadlock will occur among r_1, r_3, r_4, and r_5. If r_1 moves one step forward, r_1, r_3, and r_4 form a deadlock (c_5), while if r_4 moves forward, r_1, r_4, and r_5 form a deadlock (c_6).

There is no doubt that to ensure safety, each admissible configuration should not be a higher-order deadlock. Let $\mathcal{C}_{free} = \mathcal{C}_{dfree} \setminus \mathcal{C}_{hdead}$. In the sequel, we can give the problem statement studied in this chapter.

Problem 3. Given a multi-robot system with N robots, whose LTS models are $\{\mathcal{T}_i\}_{i\in\mathbb{I}_N}$, find an online and distributed control policy for the system such that all reachable configurations are in $\mathcal{C}_{free}$.

7.3 Higher-Order Deadlocks and Their Avoidance

In this section, we study the characteristics of higher-order deadlocks from the system level, based on which we develop a distributed method in the next subsection to detect higher-order deadlocks by each robot.

Definition 21. An edge-colored digraph is a quadruple $\langle V, E, I_c, C\rangle$, where V is a finite set of vertices, E is a finite set of edges, I_c is a finite set of colors, and $C : E \to I_c$ is a function assigning each edge with a color in I_c.

Definition 22. The edge-colored digraph produced from a multi-robot system, $\{\mathcal{T}_i = \langle S^i, \Sigma_i, \rightarrow_{i,move}\rangle, i \in \mathbb{I}_N\}$, is a quadruple $G_{\mathcal{T}} = \langle V, E, I_c, C\rangle$, where $V = \cup_{i\in\mathbb{I}_N} S^i$, $E = \cup_{i\in\mathbb{I}_N} \rightarrow_{i,move}$, $I_c = \mathbb{I}_N$ containing N different colors, and C is a color mapping satisfying $\forall e \in \rightarrow_{i,move}, C(e) = i$.

This definition means that in the graphic representation of a multi-robot system, all edges from the same robot are colored by the same color. Based on Proposition 2 in Chapter 5, we can conclude that there are no two states that are connected by two or more different colored edges in $G_{\mathcal{T}}$.

Definition 23. An individual walk of robot r_i in $G_{\mathcal{T}}$ is a sequence of vertices and edges with the form $\langle s_1, (s_1, s_2), s_2, \ldots, s_{n-1}, (s_{n-1}, s_n), s_n\rangle$, where $s_k \in S^i$ for all $k \in \mathbb{I}_n$.

Without ambiguity, w can also be simplified as $w = \langle s_1, s_2, \ldots, s_n\rangle$. The tail of w, denoted as $t(w)$, is $t(w) = s_1$, and the head of w, denoted as $h(w)$, is $h(w) = s_n$. If a robot is at a private state, then it cannot block any robot. So in the rest, we only need to study the situation that a robot is at a collision state. We have the following definition.

Definition 24. Suppose $w(c) = \langle s_1, s_2, \ldots, s_n\rangle$ is an individual walk of r_i at configuration c, then

- $w(c)$ is risky if (1) $\forall k \in \mathbb{I}_n$, $s_k \in S^i_\alpha$; and (2) r_i is at s_1, while s_n is occupied by another robot r_j at c.
- $w(c)$ is safe if (1) $\forall k \in \mathbb{I}_{n-1}, s_k \in S^i_\alpha$, and $s_n \in S^i_\beta$; and (2) r_i is at s_1 whereas other states are empty at c.

The concepts of risky and safe walks are dependent on system configurations. Given a configuration c where a robot is at a collision state, this robot has either risky walks or a safe walk. Each robot can check risky or safe walks independently since it only needs to detect its own path.

If a robot has a safe walk currently, its one-step move cannot cause collisions or deadlocks since as the last resort it can move to its private state while others stop at their current states. Hence, we only need to consider the robots that have risky walks.

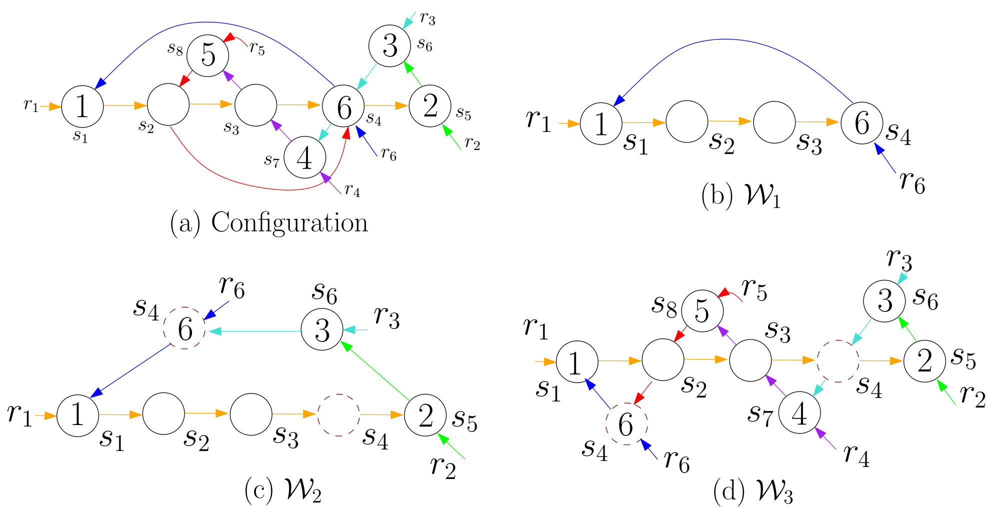

(a) Configuration (b) $\mathcal{W}_1$ (c) $\mathcal{W}_2$ (d) $\mathcal{W}_3$

FIG. 7.2: A configuration containing three circuits.

The following descriptions are related to a given configuration c even if it is not shown explicitly.

For a risky walk w, the states between its tail and head are called intermediate states of w, denoted as $S_E(w)$; the length of w is defined as $L(w) = |S_E(w)|$. Note that a risky walk with length l has $l + 2$ states, i.e., l intermediate states + one tail + one head. For convenience, we sometimes use w_{ij} to denote r_i's risky walk whose head is occupied by r_j. For example, at the configuration shown in Fig. 7.2(a), r_1 has two risky walks: $w_{16} = \langle s_1, s_2, s_3, s_4\rangle$ and $w_{12} = \langle s_1, s_2, s_3, s_4, s_5\rangle$; $S_E(w_{16}) = \{s_2, s_3\}$, $L(w_{16}) = 2$; and $S_E(w_{12}) = \{s_2, s_3, s_4\}$, $L(w_{12}) = 3$.

Definition 25. $\mathcal{W} = \langle w_1, w_2, \ldots, w_m\rangle$ is a circuit if (1) $\forall i \in \mathbb{I}_m$, w_i is a risky walk, and $h(w_i) = t(w_{i+1})$ where $w_{m+1} = w_1$, and (2) $\forall i_1, i_2 \in \mathbb{I}_m$, $i_1 \neq i_2$, w_{i_1} and w_{i_2} belong to two different robots.

For example, the configuration shown in Fig. 7.2(a) contains three circuits, shown in Figs. 7.2(b)–(d). $\mathcal{W}_1 = \langle w_{16}, w_{61}\rangle$ is a circuit since w_{16} and w_{61} are risky walks of r_1 and r_6, respectively; $h(w_{16}) = s_4 = t(w_{61})$ and $t(w_{16}) = s_1 = h(w_{61})$. Similarly, $\mathcal{W}_2 = \langle w_{12}, w_{23}, w_{36}, w_{61}\rangle$, and $\mathcal{W}_3 = \langle w_{12}, w_{23}, w_{34}, w_{45}, w_{56}, w_{61}\rangle$ are circuits too.

For a circuit $\mathcal{W}$, the following notations are used in the sequel. $\mathcal{I}(\mathcal{W}) = \{i_1, i_2, \ldots, i_m\}$ is the set of indices of robots in $\mathcal{W}$, where r_{i_j} is at $t(w_j)$. $S_E(\mathcal{W}) = \cup_{j=1}^{m} S_E(w_j)$ is the set of intermediate states of $\mathcal{W}$. $S_\alpha(\mathcal{W}) = \cup_{i_j, i_k \in \mathcal{I}(\mathcal{W}) \wedge i_j \neq i_k}\ S_\alpha^{i_j} \cap S_\alpha^{i_k}$ denotes

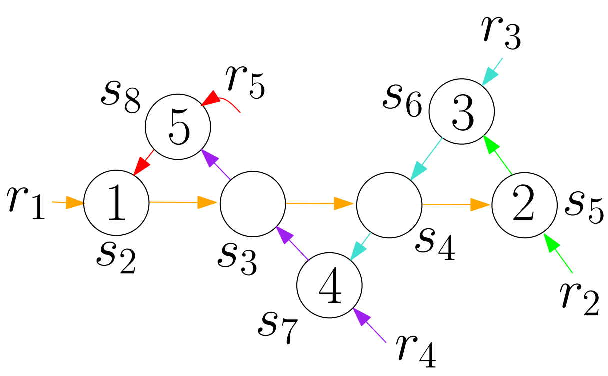


FIG. 7.3: The sub-circuit $\mathcal{W}_3'$ of $\mathcal{W}_3$ given in Fig. 7.2.

all collision states between any two different robots in $\mathcal{I}(\mathcal{W})$, and $\mathcal{I}_i(\mathcal{W})$ is the set of robots at the states of $S_E(\mathcal{W})$.

Next, we introduce the sub-circuit of a circuit. Suppose $\mathcal{W} = \langle w_1, w_2, \ldots, w_m\rangle$ is a circuit, where $w_j = \langle s_1^j, s_2^j, \ldots, s_{k_j}^j\rangle$ is a risky walk of r_{i_j}, $\forall j \in \mathbb{I}_m$. Select a movable robot r_{i_p}, $p \in \mathbb{I}_m$, and let it move one step forward. The risky walk w_p changes to $w_p' = \langle s_2^p, \ldots, s_{k_p}^p\rangle$. Then verification is done to check whether there still exists a circuit with some of the risky walks in $\mathcal{W}$. The process starts from r_{i_p} with w_p'. First, r_{i_p} sends the information of s_2^p to $r_{i_{p+1}}$. After it receives this message, $r_{i_{p+1}}$ checks whether w_{p+1} passes over s_2^p. If not, the message is again sent to $r_{i_{p+2}}$ by $r_{i_{p+1}}$. Then $r_{i_{p+2}}$ begins to check w_{p+2}. Repeat sending the message to robots until a robot, say $r_{i_{p_1}}$, checks that its risky walk, w_{p_1}, passes over s_2^p. Note that $w_j = w_{j-m}$ if $j > m$. Thus, $w_{p_1} = \langle s_1^{p_1}, s_2^{p_1}, \ldots, s_2^p, \ldots, s_{k_{p_1}}^{p_1}\rangle$ changes to $w_{p_1}' = \langle s_1^{p_1}, s_2^{p_1}, \ldots, s_2^p\rangle$, and $\mathcal{W}' = \langle w_p', w_{p+1}, \ldots, w_{p_1-1}, w_{p_1}'\rangle$ is a circuit. We call $\mathcal{W}'$ sub-circuit of $\mathcal{W}$. Note the risky walks between w_p' and w_{p_1}' in $\mathcal{W}'$ are the same as those in $\mathcal{W}$, while w_p' and w_{p_1}' are parts of w_p and w_{p_1}, respectively.

For example, consider $\mathcal{W}_3 = \langle w_{12}, w_{23}, w_{34}, w_{45}, w_{56}, w_{61}\rangle$ in Fig. 7.3. Let r_1 move to s_2. Then $w_{12}' = \langle s_2, s_3, s_4, s_5\rangle$, the risky walk of r_5 becomes $w_{51} = \langle s_8, s_2\rangle$, and r_6 is excluded from $\mathcal{W}_3$. Thus, $\mathcal{W}_3' = \langle w_{12}', w_{23}, w_{34}, w_{45}, w_{51}\rangle$ is a sub-circuit of $\mathcal{W}_3$.

A circuit $\mathcal{W}$ may contain other smaller circuits. The difference between the smaller circuits and sub-circuits is that the smaller circuits must coexist with $\mathcal{W}$ at the same configuration c, while sub-circuits are generated by the moves of some robots in $\mathcal{W}$ and are existent at a succeeding configuration of c. For example, as shown in Fig. 7.2, $\mathcal{W}_1$

and $\mathcal{W}_2$ are two smaller circuits of $\mathcal{W}_3$ at the current configuration, but they are not sub-circuits of $\mathcal{W}_3$; while $\mathcal{W}_3'$ is a sub-circuit at the succeeding configuration of that in Fig. 7.2(a).

In the sequel, we describe the relation between deadlocks and circuits.

Definition 26. A deadlock cycle is a circuit where the length of each risky walk is 0.

Proposition 3. A deadlock cycle contains at least 3 robots.

Proof. First, consider there exists a deadlock cycle with two robots. Without loss of generality, suppose $\mathcal{W} = \langle w_1, w_2 \rangle$. We have $h(w_1) = t(w_2)$ and $h(w_2) = t(w_1)$, implying that there are two vertices connected by two colors. This violates Proposition 2. Second, as shown in Fig. 7.1(a), there exists a deadlock cycle with three robots. □

Proposition 4. A configuration c is a deadlock configuration if and only if there exists a deadlock cycle in c.

Proof. Suppose c is a deadlock configuration satisfying $Pos_{i_m}(c(i_m)) = c(i_{m+1})$ for $m = 1, 2, \ldots, k$ and $i_{k+1} = i_1$. Thus, r_{i_m} has a risky walk $w_{i_m} = \langle c(i_m), c(i_{m+1}) \rangle$. Hence, $\mathcal{W} = \langle w_{i_1}, w_{i_2}, \ldots, w_{i_m} \rangle$ is a deadlock cycle.

Suppose $\mathcal{W} = \langle w_{i_1}, w_{i_2}, \ldots, w_{i_k} \rangle$ is a deadlock cycle at c. $\forall m \in \{1, 2, \ldots, k\}$, since the length of w_{i_m} is 0, $w_{i_m} = \langle s_{i_m}, s_{i_{m+1}} \rangle$, where s_{i_m} and $s_{i_{m+1}}$ are the current states of r_{i_m} and $r_{i_{m+1}}$, respectively; and $i_{k+1} = i_1$. So $Pos_{i_m}(s_{i_m}) = s_{i_{m+1}}$. This means $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$ satisfy Definition 19. Hence, c is a deadlok configuration. □

As described before, it is difficult to predict deadlocks in advance owing to the existence of higher-order deadlocks. So we first study the characteristics of higher-order deadlocks.

Definition 27. A circuit $\mathcal{W}$ is called a k-th order deadlock if (1) for any movable robot, its one-step move causes a lower-order deadlock with these robots, and (2) there exists a robot such that its one-step move causes a $(k-1)$-th order deadlock with these robots.

Remark 6. We assume that a suitable local continuous controller is available for each robot that takes into account the robot's dynamics and can stop the robot in a short time. One-step move corresponds to the switch from the current segment to the next one physically. It takes place only at the end of the current segment.

Remark 7. If a circuit contains a smaller circuit which is a higher-order deadlock, the larger one is also a higher-order deadlock. It makes no sense to study the larger circuit without resolving the contained higher-order deadlock. So we focus on simple higher-order deadlocks, meaning that the smaller circuits in a higher-order deadlock are always deadlock-free.

Indeed, a k-th order deadlock is a circuit $\mathcal{W}$ such that a deadlock occurs inevitably within k times of transitions. Here the number of transitions is counted by the moves of robots that are involved in the sub-circuits of $\mathcal{W}$ before their moves. A deadlock cycle is also called 0-th order deadlock.

Intuitively, a k-th order deadlock occurs because the robots in the circuit $\mathcal{W}$, i.e., r_{i_j}, $i_j \in \mathcal{I}(\mathcal{W})$, are in a "circular wait" in order to avoid lower-order deadlocks or collisions. In other words, r_{i_1} cannot move forward because its move can cause a lower-order deadlock. Thus, it has to wait for the robot in its path, say r_{i_2}, to move away. However, r_{i_2} cannot move forward since its move can also cause a lower-order deadlock. So r_{i_2} also needs to wait for the move of the robot in its path, say r_{i_3}. This process iterates until r_{i_m} needs to wait for the move of r_{i_1}. Thus, a circular wait occurs and none of them can move. Note, collision avoidance leads to deadlock cycles while the $(k-1)$-th order deadlock avoidance leads to the k-th order deadlock.

For example, as shown in Fig. 7.1, c_0 is a second-order deadlock. A deadlock can happen after one-step move (c_3) or two-step move ($c_4 - c_6$). Note that as described before, the number of steps of move is counted by the robots that are always in the resulting sub-circuits. Hence, at c_1, since r_5 is not in c_0's sub-circuit resulting from the move of r_1, its motion is not taken into consideration during the evolution of c_1. Similarly, at c_2, the motion of r_2 is not taken into consideration in the evolution of c_2.

Lemma 1. If $\mathcal{W}$ is a higher-order deadlock, $S_E(\mathcal{W}) \subseteq S_\alpha(\mathcal{W})$.

Proof. Suppose $\mathcal{W} = \langle w_1, w_2, \ldots, w_m \rangle$ is a higher-order deadlock, where w_j is a risky walk of r_{i_j}, $\forall j \in \mathbb{I}_m$. If there exists s, $s \in S_E(w_k)$, $k \in \mathbb{I}_m$, such that $s \notin S_\alpha(\mathcal{W})$. Then,

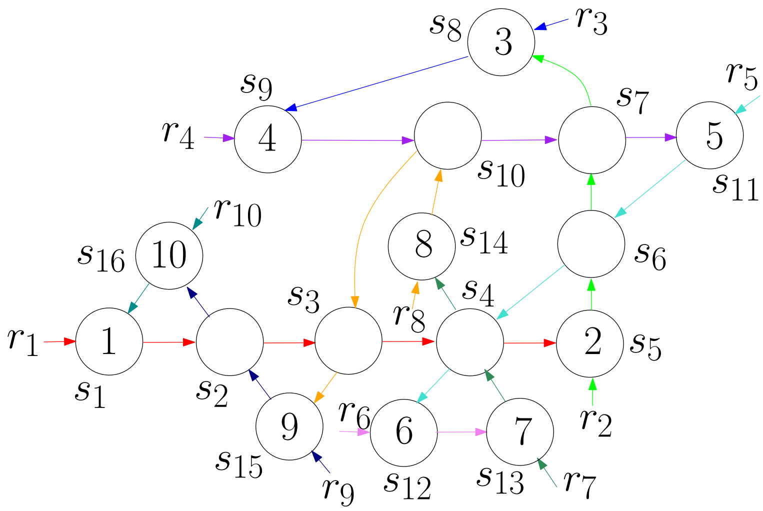


FIG. 7.4: An example of a live circuit with 10 robots. The states with numbers are the current states of the corresponding robots.

we can conclude that $\mathcal{W}$ is live. First, there exists an execution such that the robots between r_{i_k}'s current state and s can move away from w_k without causing any deadlocks except those with r_{i_k} and the robot at $h(w_k)$. Otherwise, there exists a lower-order deadlock in $\mathcal{W}$, which is in conflict with the simple higher-order deadlock assumption. Then, let r_{i_k} move to s. However, after it moves to s, r_{i_k} cannot block the motion of other robots in $\mathcal{W}$ because of $s \notin S_\alpha(\mathcal{W})$. This means there is no "circular wait" among the robots in $\mathcal{W}$ anymore. Thus, $\mathcal{W}$ is live. This is a contradiction. Hence, $\forall s \in S_E(\mathcal{W})$, $s \in S_\alpha(\mathcal{W})$, i.e., $S_E(\mathcal{W}) \subseteq S_\alpha(\mathcal{W})$. □

Note that Lemma 1 describes a necessary but not necessarily sufficient condition. For example, robots $r_1 - r_{10}$ in Fig. 7.4 form a circuit $\mathcal{W} = \langle w_{12}, w_{23}, w_{34}, w_{45}, w_{56}, w_{67}, w_{78}, w_{89}, w_{9,10}, w_{10,1}\rangle$, where $w_{12} = \langle s_1, s_2, s_3, s_4, s_5\rangle$, $w_{23} = \langle s_5, s_6, s_7, s_8\rangle$, $w_{34} = \langle s_8, s_9\rangle$, $w_{45} = \langle s_9, s_{10}, s_7, s_{11}\rangle$, $w_{56} = \langle s_{11}, s_6, s_4, s_{12}\rangle$, $w_{67} = \langle s_{12}, s_{13}\rangle$, $w_{78} = \langle s_{13}, s_4, s_{14}\rangle$, $w_{89} = \langle s_{14}, s_{10}, s_3, s_{15}\rangle$, $w_{9,10} = \langle s_{15}, s_2, s_{16}\rangle$, and $w_{10,1} = \langle s_{16}, s_1\rangle$. Clearly, $\mathcal{W}$ satisfies $S_E(\mathcal{W}) \subseteq S_\alpha(\mathcal{W})$. But it is live. Indeed, let r_2 move to s_7 first. Thus, r_1 can move to s_5 and then to its private state. Second, r_{10}, r_9, r_8, r_7, r_6, and r_5 can move to their own private states in turns. Third, r_4 moves to s_{10}. So r_3 and r_2 can move to their private states in sequence. At last, r_4 can move to its private state. Thus, no deadlocks can occur during their motion in this circuit.

Lemma 2. If $\mathcal{W} = \langle w_1, w_2, \ldots, w_m\rangle$ is a k-th order deadlock, $k \leq m - 3$.

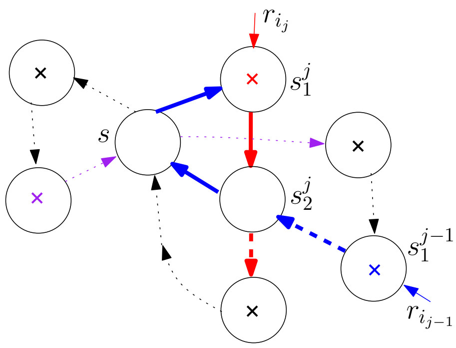


FIG. 7.5: The bold edges are the $move$ transitions of r_{i_j} and $r_{i_{j-1}}$, the states with crosses denote the current states of robots, and the dotted ones represent the $move$ transitions of the rest robots in $\mathcal{W}$.

Proof. For any movable robot r_{i_j}, its one-step move will release at least the robot $r_{i_{j-1}}$ from $\mathcal{W}$ [$\triangleq T_1$]. (For example, as the system shown in Fig. 7.2, when r_1 moves to s_2 from s_1, r_6 is excluded from the sub-circuit.) We prove T_1 using proof by contradiction.

Suppose $r_{i_{j-1}}$ is still in the sub-circuit of $\mathcal{W}$ after r_{i_j} moves one-step forward [$\triangleq \neg T_1$]. Then, let $w_j = \langle s_1^j,\, s_2^j,\, \ldots,\, s_{k_j}^j \rangle$ be the risky walk of r_{i_j}. w_{j-1} in $\mathcal{W}$ should satisfy: (1) $s_2^j,\, s_1^j \in w_{j-1}$; (2) there exists at least one state between s_2^j and s_1^j; and (3) only r_{i_j} and $r_{i_{j-1}}$ can traverse s_2^j (Otherwise a sub-circuit is constructed before $r_{i_{j-1}}$ is considered). Without loss of generality, suppose $w_{j-1} = \langle s_1^{j-1},\, \ldots,\, s_{k_{j-1}}^{j-1},\, s_2^j,\, s,\, s_1^j \rangle$. Thus, $\mathcal{W}$ is with the structure shown in Fig. 7.5.

Since $\mathcal{W}$ is a higher-order deadlock, there exists an execution such that r_{i_j} and $r_{i_{j-1}}$ still form a higher-order deadlock with the robots in this circuit and the states between r_{i_j} and $r_{i_{j-1}}$ are idle. There are two cases after such an execution. The first one is that r_{i_j} does not need to move one step forward and the second one is that r_{i_j} needs to move one step forward. For the first one, let $r_{i_{j-1}}$ move to s, then r_{i_j} to s_2^j. So r_i does not block any other robot. This means there is no circular wait anymore. Thus, there is no deadlock. (For example, as shown in Fig. 7.6(a), let r_5 ($r_{i_{j-1}}$) move to s, then r_1 (r_{i_j}) can move to s_2^1. Next, r_5 can move to s_1^1. Thus, r_5, r_4, r_3, and r_2 can leave this circuit, and finally r_1 can also leave this circuit. So there is no deadlock.) However, $\mathcal{W}$ is a higher-order deadlock. So in this case, $\neg T_1$ is not satisfied and thus T_1 is satisfied. For the second case, it means that r_{i_j} blocks some robots' motion even though r_{i_j} is not the robot that they need to wait for leaving in the circular wait. Thus, when r_{i_j} moves to s_2^j, other robots can move sequentially and finally all robots at the states of $r_{i_{j-1}}$ can move

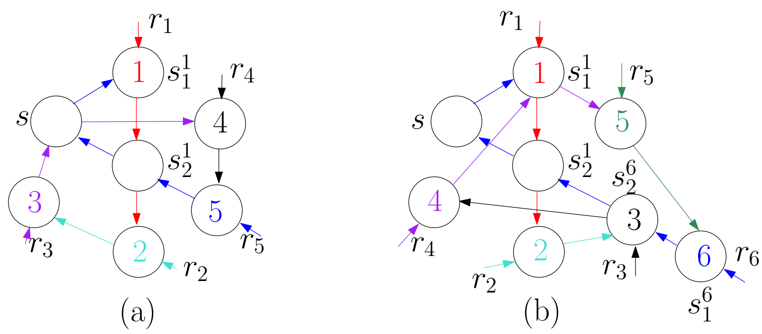


FIG. 7.6: Examples to illustrate the proof of Lemma 2.

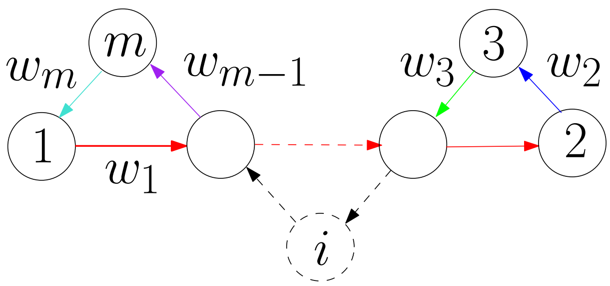


FIG. 7.7: An example of an $(m-3)$-th order deadlock containing m robots.

away from their current states. Then, the robots that need to wait for the leaving of these robots can now leave the circuit. Thus, $r_{i_{j_1}}$ can move forward and does not block other robots. This means $r_{i_{j-1}}$ cannot form a deadlock with r_{i_j}. (For example, as shown in Fig. 7.6(b), after r_1 (r_{i_j}) moves to s_2^1, r_4 can move to s_1^1, causing that r_3 can leave s_2^6. Thus, r_2 can leave the circuit. So r_6 ($r_{i_{j-1}}$) can move to s_2^6 and does not block others.) However, if $\neg T_1$ is satisfied, $r_{i_{j-1}}$ should form a deadlock with r_{i_j}. Thus, $\neg T_1$ cannot be satisfied and T_1 is satisfied.

Thus, we prove that the first statement, i.e., T_1, is satisfied. Repeatedly applying T_1, we can conclude that a k-step move can release at least k robots. After k-step moves, there are still some robots that form a deadlock cycle. The number is not less than 3 based on Proposition 3. Thus, the total number of robots is not less than $k+3$, i.e., $m \geq k+3$. Hence, $k \leq m-3$.

On the other hand, there exists an $(m-3)$-th order deadlock with m robots. Consider the circuit shown in Fig. 7.7. $\mathcal{W} = \langle w_1, w_2, \ldots, w_m \rangle$, where (1) $\forall i \in \mathbb{I}_m$, $S_E(w_i) \subseteq S_E(w_1)$; and (2) $|S_E(w_1)| = k$, $|S_E(w_2)| = |S_E(w_m)| = 0$, and $|S_E(w_j)| = 1$ for other risky walks. Clearly, $k = m-3$. Moreover, a deadlock cycle can occur inevitably within $m-3$ steps. Hence, $\mathcal{W}$ is an $(m-3)$-th order deadlock. $\square$

Lemma 3. If $\mathcal{W}$ is a k-th order deadlock, $|S_E(\mathcal{W})| = k$.

Proof. On one hand, during the motion to form a deadlock cycle, each robot moves along the intermediate states of its risky walk in $\mathcal{W}$. Besides, based on the definition of sub-circuit, once it has been traversed, an intermediate state is excluded from the sub-circuit. Thus, each intermediate state can be traversed at most one time during the process to form a deadlock cycle. Hence, $|S_E(\mathcal{W})| \geq k$. On the other hand, when some robots in $\mathcal{W}$ form a deadlock cycle after some steps of move, they are at the intermediate states of the original risky walks in $\mathcal{W}$. Thus, for any higher-order deadlock, a deadlock cycle must occur when all the intermediate states are traversed. Hence, $|S_E(\mathcal{W})| \leq k$. From the above analysis, $|S_E(\mathcal{W})| = k$. □

According to Lemmas 2 and 3, we can get the boundary of the number of intermediate states in a higher-order deadlock.

Lemma 4. For a higher-order deadlock $\mathcal{W} = \langle w_1, w_2, \ldots, w_m \rangle$, $|S_E(\mathcal{W})| \leq m - 3$. Moreover, for all $i \in \mathbb{I}_m$, $|S_E(w_i)| \leq m - 3$.

Based on above lemmas, the following statement gives criteria to check whether a circuit is a higher-order deadlock.

Theorem 7. Suppose $\mathcal{W} = \langle w_1, w_2, \ldots, w_m \rangle$ is a circuit and $|S_E(\mathcal{W})| = k$. $\mathcal{W}$ is live if (1) $k > m - 3$ or $S_E(\mathcal{W}) \setminus S_\alpha(\mathcal{W}) \neq \emptyset$, or (2) there exists a movable robot such that its one-step move either causes no circuits or forms a sub-circuit $\mathcal{W}'$ satisfying $|S_E(\mathcal{W}')| > |\mathcal{I}(\mathcal{W}')| - 3$ or $S_E(\mathcal{W}') \setminus S_\alpha(\mathcal{W}') \neq \emptyset$.

Proof. If $k > m - 3$, $\mathcal{W}$ is deadlock-free based on the converse-negative proposition of Lemma 4; while if $S_E(\mathcal{W}) \setminus S_\alpha(\mathcal{W}) \neq \emptyset$, based on the converse-negative proposition of Lemma 1, $\mathcal{W}$ is deadlock-free. Thus, if condition (1) is satisfied, $\mathcal{W}$ is live.

Suppose condition (2) is satisfied. If the one-step move of a robot causes no circuit among these robots, these robots cannot be in a deadlock. Thus, $\mathcal{W}$ is deadlock-free. Otherwise, if the resulting sub-circuit $\mathcal{W}'$ satisfies one of the two described conditions, we can conclude $\mathcal{W}'$ is live based on the proof of condition (1). Thus, $\mathcal{W}$ is live. □

Remark 8. If the computation time is available, the second condition can be extended to the following one: There exists a movable robot such that its finite-step move either causes no circuits or forms a sub-circuit $\mathcal{W}'$ satisfying $|S_E(\mathcal{W}')| > |\mathcal{I}(\mathcal{W}')| - 3$ or $S_E(\mathcal{W}') \setminus S_\alpha(\mathcal{W}') \neq \emptyset$.

In the sequel, we describe the procedure for higher-order deadlock checking. The details are shown in Algorithm 6. In the algorithm, A_i is an N-dimensional vector, where $A_i(j) = l < \infty$ means that there exists a risky walk of r_i such that the head is occupied by r_j and the length is l; $A_i(j) = \infty$ means the current state of r_j is not in S^i or there exists at least one private state of r_i between r_i and r_j; $\mathbb{W}_i$ collects the risky walks that r_i can construct if r_i is at s. Note that each robot computes A_i and $\mathbb{W}_i$ independently.

The algorithm contains three steps. The first one is that r_i searches for its next $N-2$ states and updates A_i and $\mathbb{W}_i$, i.e., Lines 1$-$7 in Algorithm 6. The second step is that r_i searches for all circuits it can form by communicating with others, i.e., Lines 8$-$15 in Algorithm 6. In this step, for each robot r_j at its next $N-2$ states, r_i sends message $(r_j, \langle r_i, s\rangle, RW, Index)$ to r_j. The first parameter in the message identifies the receiver of the message; the second one denotes the robot activating the procedure of deadlock detection and its checked state; RW collects the risky walks delivered by the previous robots; and $Index$ is the set of remaining robots and it guarantees that different risky walks in the generated circuit are of different robots. Once it receives $(r_j, \langle r_i, s\rangle, RW, Index)$, r_j executes its $visit$ function, taking the received message as an input.

The detailed procedure of a robot's $visit$ function is given in Function $visit$. Suppose r_j receives a message $(r_j, \langle r_i, s\rangle, RW, Index)$ from $r_{j'}$. r_j first detects robots at its next $N-2$ states, stored in N_j (Lines 3 $-$ 9 in Function $visit$). If $N_j = \emptyset$, meaning that there are no circuits with r_j, so r_j sends $RW_f = \emptyset$ back to $r_{j'}$ (Lines 11 and 12 in Function $visit$). Otherwise, for each robot r_k in N_j, r_j adds its risky walk w_{ik} to RW_f (Line 15 in Function $visit$). If r_k is r_i, then this branch is finished and r_j sends the generated RW_f back to $r_{j'}$ (Lines 16 and 17 in Function $visit$); otherwise, r_j sends a message $(r_k, \langle r_i, s\rangle, RW_f, Index)$ to r_k, and waits for r_k's response (Line 19 in

Algorithm 6: Process of higher-order deadlock detection for r_i if it is at s.

Input : r_i's LTS model $\mathcal{T}_i$; the state to be checked: s.
Output: Boolean value.

/* Step 1: Search for the risky walks. */
$p = s$; $q = Pos_i(p)$; $w = \langle p, q\rangle$; $\mathbb{W}_i = \emptyset$; $A_i = \infty$;
for $k = 1$ **to** $N - 2$ **do**
  if $q \in S^i_\beta$ **then**
    /* Reach a private state. */
    Break;
  else if *r_i detects another robot, say r_j, at q* **then**
    $A_i(j) = k - 1$; $w_{ij} = w$; $\mathbb{W}_i = \mathbb{W}_i \cup \{w_{ij}\}$;
  $p = q$; $q = Pos_i(p)$; $w = \langle w, q\rangle$;
/* Step 2: Search for the circuits that r_i can form. */
$Circuit = \emptyset$; /* Collect circuits. */
$N_i = \{j : A_i(j) \neq \infty$ and $j \neq i\}$;
/* Detect the neighbors on r_i's path */
$Index = \mathbb{I}_N$;
for *each* $j \in N_i$ **do**
  $RW = \emptyset$; /* RW stores the risky walks forming a circuit. */
  $RW = \langle w_{ij}\rangle$;
  Send message $(r_j, \langle r_i, s\rangle, RW, Index)$ to r_j and wait for response from r_j's $visit$ function;
$Circuit$ = All returned circuits RW_f from other robots;
/* Step 3: Higher-order Deadlock Checking. */
if $Circuit == \emptyset$ **then**
  return true;
while $Circuit \neq \emptyset$ **do**
  Select the first circuit, say $\mathcal{W}$, in $Circuit$;
  $Circuit = Circuit \setminus \{\mathcal{W}\}$;
  Broadcast $\mathcal{W}$ to the system;
  Count $S_E(\mathcal{W})$, $\mathcal{I}(\mathcal{W})$, and $S_\alpha(\mathcal{W})$;
  if $(|S_E(\mathcal{W})| > |\mathcal{I}(\mathcal{W})| - 3) \vee (S_E(\mathcal{W}) \setminus S_\alpha(\mathcal{W}) \neq \emptyset)$ **then**
    /* Satisfy the first condition in Theorem 7. */
    return true;
  else
    Call each robot execute its *check*$(\mathcal{W}, r_j)$;
    return $\vee_{j\in\mathcal{I}(\mathcal{W})} check(\mathcal{W}, r_j)$;

Function $visit(r_j, \langle r_i, s\rangle, RW, Index)$

/* r_j adds a risky walk to RW and then sends the message to the next robot. */

Input: Received message (r_j, $\langle r_i, s\rangle$, RW, $Index$) from $r_{j'}$.

1 $RW_1 = RW$;
2 $p = s^j_{cur}$; $q = Pos_j(p0$; $w = \langle p, q\rangle$; $\mathbb{W}_j = \emptyset$; $A_j = \infty$;
3 **for** $k = 1$ **to** $N - 2$ **do**
  /* r_j checks its next $N - 2$ states. */
4   **if** $q \in S^j_\beta$ **then**
5   Break;
6   **else if** *r_j detects a robot, say r_l, at q* **then**
7     $A_j(l) = k - 1$; $w_{jl} = w$; $\mathbb{W}_j = \mathbb{W}_j \cup \{w_{jl}\}$;
8   $p = q$; $q = Pos_j(p)$; $w = \langle w, q\rangle$;
9 $N_j = \{k : k \in Index_1$ and $A_j(k) \neq \infty\}$;
10 $Index_1 = Index \setminus \{j\}$;
11 **if** $N_j == \emptyset$ **then**
  /* No circuits with respect to r_i. */
12   $RW_f = \emptyset$ and send it back to $r_{j'}$;
13 **else**
14   **for** *each* $k \in N_j$ **do**
15     $RW_f = \langle RW_1, w_{jk}\rangle$;/* Add a risky walk to RW_1. */
16   **if** $r_k == r_i$ **then**
17   Send RW_f to $r_{j'}$;
18   **else**
19   Send message $(r_k, \langle r_i, s\rangle, RW_f, Index_1)$ to r_k and wait for response;
20   Once receive RW_f from r_k, transmit it to $r_{j'}$;

Function $visit$); when it receives response with RW_f from r_k, r_j further transmits this response to $r_{j'}$ (Line 20 in Function $visit$).

The third step of Algorithm 6 is to check the circuits, i.e., Lines 16–27 in Algorithm 6. Function $check(\mathcal{W}, r_{i_j})$ in this step is used to check whether r_{i_j}'s move satisfies the second condition in Theorem 7, where $\mathcal{W} = \langle w_1, \ldots, w_m\rangle$ is a circuit and $w_j = \langle s^j_1, s^j_2, \ldots, s^j_{m_j}\rangle$ is a risky walk of r_{i_j}. Each robot can execute this function independently once it receives the corresponding circuit $\mathcal{W}$.

Next, we give the complexity analysis of the deadlock detection process. Based on Algorithm 6, there are three steps in this process. The first step is to check the status of the next $N - 2$ states. It can be done in $O(N)$ time. The second one is to search for the

Function *check*($\mathcal{W}$, r_{i_j})

$w'_j = \langle s^j_2, \ldots, s^j_{m_j} \rangle$; $\mathcal{W}' = \langle w'_j \rangle$;
/* Search for a sub-circuit of $\mathcal{W}$ with w'_j. */
for $k = j + 1$ **to** $j + m - 2$ **do**
    if $k > m$ **then**
        $w_k = w_{k-m}$;
    if $s^j_2 \in w_k$ **then**
        /* A sub-circuit is formed. */
        $w'_k = \langle s^k_1, \ldots, s^j_2 \rangle$; $\mathcal{W}' = \langle \mathcal{W}', w'_k \rangle$;
        Break;
    else
        $\mathcal{W}' = \langle \mathcal{W}', w_k \rangle$;
if $k == j + m - 1$ **then**
    return true;
else
    Count $S_E(\mathcal{W}')$, $\mathcal{I}(\mathcal{W}')$, and $S_\alpha(\mathcal{W}')$;
    if $(|S_E(\mathcal{W}')| > |\mathcal{I}(\mathcal{W}')| - 3) \vee (S_E(\mathcal{W}') \setminus S_\alpha(\mathcal{W}') \neq \emptyset)$ **then**
        /* The second condition in Theorem 7 is satisfied. */
        return true;
    else
        return false;

circuits by communicating with other robots. Note that each robot can send messages to different robots at the same time and the robots receiving the messages can check their own states simultaneously. Thus, the main computation for each robot is to check its next $N - 2$ states. Hence, the computation complexity for this step is $O(N^2)$. The third step is to verify whether there are deadlocks in the received circuits. Suppose the number of detected circuits is M, then the computation complexity is $O(M)$. Note that in theory, $M = O(2^N)$ for general cases. However, in our method, each robot only needs to look ahead at most $N - 1$ states, thus, with our discretization, the number of circuits should not be large. Moreover, in practice, the number of robots in a multi-robot system cannot be too large or change greatly. Thus, our method cannot cause high complexity and can work well in practice.

Algorithm 7 gives the collision and deadlock avoidance algorithm for robot r_i. The procedure can be described as follows. Before it moves to the next state, r_i needs

Algorithm 7: Collision and deadlock avoidance of r_i.

Input : Robot r_i's LTS model $\mathcal{T}_i$.
Output: Motion without collisions and deadlocks of r_i.

1 $s_{cur} = c_{cur}(i)$, $s_{next} = Pos_i(s_{cur})$, the negotiation region X;
2 **if** $s_{next} \in S^i_\beta$ **then**
    /* The succeeding state is a private one. */
3   Execute the transition $s_{cur} \xrightarrow{move}_i s_{next}$;
4   **if** $s_{cur} \in S^i_\alpha$ **then**
5         $Sign(s_{cur}) = 0$;
6     $c_{cur}(i) = s_{next}$;
7 **else if** $Sign(s_{next}) == 1$ **then**
8   Stop for a given delay and re-perform the algorithm;
9 **else**
10   **if** *Algorithm 6*(r_i, s_{next}) **then**
        /* One-step move cannot cause any higher-order deadlocks. */
11   Determine the negotiation robots E_X;
12   **if** $NEG(E_X) == r_i$ **then**
13   Execute the transition $s_{cur} \xrightarrow{move}_i s_{next}$;
14             $E_X = \emptyset$;
15   **if** $s_{cur} \in S^i_\alpha$ **then**
16                 $Sign(s_{cur}) = 0$;
17             $c_{next}(i) = s_{next}$; $Sign(s_{next}) = 1$;
18   **else**
19   Stop for a given delay and re-perform the algorithm;

to check whether its transition is allowed. Lines 2–8 are the procedure for collision avoidance. If its next state is a private state, r_i moves forward and updates its current state (Lines 2–6). If the next state is occupied by a robot, r_i cannot move forward in order to avoid collision and it tries a new attempt after a given delay (Lines 7–8). If collision avoidance is guaranteed, r_i checks whether its move can cause any higher-order deadlocks and then determine the event to be triggered (Lines 9–19). If its move cannot cause higher-order deadlocks and it wins the negotiation, r_i moves one step forward; otherwise, it re-executes the algorithm after a given delay.

According to Algorithm 7, deadlock detection is to check whether there exists a higher-order deadlock when the robot is at its succeeding state. Thus, each robot should look ahead $N - 1$ states, two endpoints, including the succeeding state, and $N - 3$

intermediate states, to determine whether it can move forward.

Theorem 8 states the correctness of Algorithm 7.

Theorem 8. Under the control of Algorithm 7, the system is always live if the initial configuration c_0 is live.

Proof. On one hand, given a deadlock-free configuration, the new generated configuration by any robot executing its Algorithm 7 is deadlock free. This is because the search process in Algorithm 6 makes sure that all circuits are searched. Thus, Lines 9−19 in Algorithm 7 guarantee that the move of a movable robot cannot cause any higher-order deadlocks. Hence, the generated configuration is live. Since c_0 is live, all reachable configurations generated by Algorithm 7 are deadlock-free. On the other hand, we assert if its one-step move can cause a higher-order deadlock currently with a set of robots, r_i first stops at its current state but can eventually move forward in the future. This is because the involved robots can move forward at least one by one due to the stop of r_i, eventually allowing r_i to move forward. Therefore, the system is live and each robot can move persistently. □

7.4 Distributive Analysis

In this section, we specify the distributed nature of the proposed algorithm.

According to Algorithms 6 and 7, to execute its local algorithms, each robot may need to (1) check the status of its collision states via on-board sensors, and (2) communicate with its neighbors to check higher-order deadlocks.

On one hand, each robot needs to retrieve the status of its own collision states, i.e., $Sign(s)$, using its on-board sensors. Indeed, during the implementation, $Sign$ is a set of independent local resources, i.e., $\{Sign(s) : s \in S_\alpha\}$. Each robot stores the local signals $\{Sign(s) : s \in S_\alpha^i\}$. To retrieve these values, the robot uses its sensors to detect whether there are some robots at its collision states within its sensing range. After a robot arrives at a collision state, other robots may use their own sensors to detect the status of this state and then can determine their motion. In this way, each

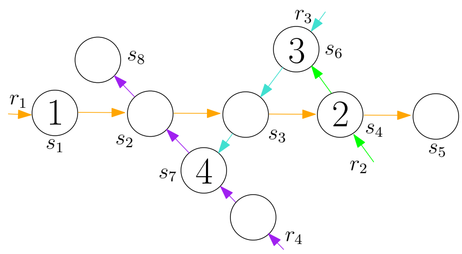


FIG. 7.8: A system with four robots that are traversing a collision region.

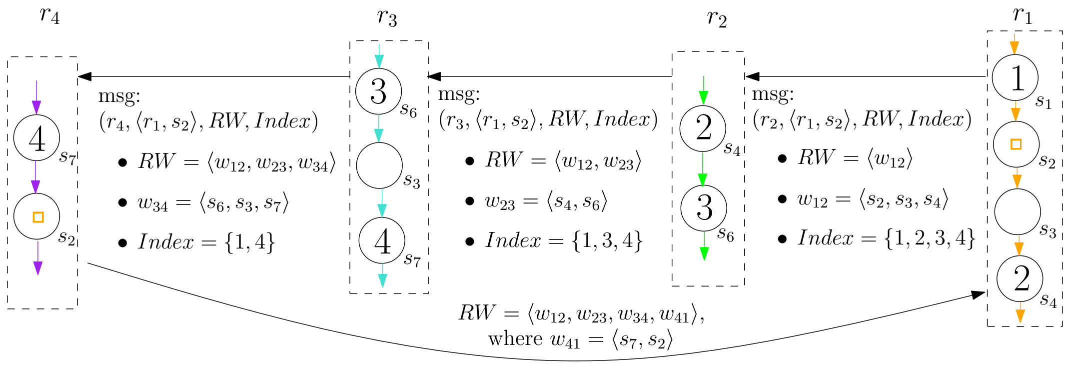


FIG. 7.9: The communication of r_1 with other robots for the deadlock checking process. s_2 is the state that r_1 needs to check.

robot manages its own local signals independently, rather than depends on external computers or other devices. For example, as shown in Fig. 7.8, the collection of local signal resources is $\{Sign(s_2)$, $Sign(s_3)$, $Sign(s_4)$, $Sign(s_6)$, and $Sign(s_7)\}$. r_1 stores $\{Sign(s_2), Sign(s_3), Sign(s_4)\}$; r_2 stores $\{Sign(s_4), Sign(s_6)\}$; r_3 stores $\{Sign(s_3)$, $Sign(s_6)$, $Sign(s_7)\}$; and r_4 stores $\{Sign(s_2), Sign(s_7)\}$. When the system is at the current configuration, r_1 retrieves $Sign(s_2) = 0$, $Sign(s_3) = 0$, and $Sign(s_4) = 1$. After r_1 moves to s_2, if r_4 starts its detection, then it could detect r_1 and set $Sign(s_2)$ stored in it to 1 using its own sensors.

Even though each robot stores its own local signals of its collision states, there is no need to synchronize the values of the same collision state stored in different robots. This means when a robot changes the value of a collision state, it does not affect the signals related to this state but stored in other robots. Indeed, suppose a robot changes one of its signals, say $Sign(s)$. If other robots need to check s to make decisions, they should first use their sensors to check the status of s and update the corresponding value, rather than the previous value.

On the other hand, a robot may communicate with its neighbors for deadlock detection. For example, consider the system shown in Fig. 7.8. Suppose at the current configuration of the system, r_1 activates the process of deadlock detection, i.e., Line 10 in Algorithm 7. To check whether its motion to s_2 can cause a higher-order deadlock, r_1 needs to obtain the circuits it can build if it is at s_2 via communication among robots. The detailed communication process is shown in Fig. 7.9. First, r_1 identifies r_2 and thus sends a message $(r_2, \langle r_1, s_2 \rangle, RW, Index)$ to r_2. When it receives this message, r_2 invokes function $visit$, adds a risky walk w_{23} into RW, and delivers the message to r_3. Similarly, r_3 and r_4 sequentially receive this message and invoke their $visit$ functions. Finally, r_4 sends the circuit $RW = \langle w_{12}, w_{23}, w_{34}, w_{41} \rangle$ back to r_1. Thus, the communication is finished and r_1 begins to check whether RW is a higher-order deadlock.

Finally, as described in Chapter 6, a robot also needs to communicate with the robots in E_X to execute the negotiation process for mutual exclusion. This process may be done by a local coordinator, which is selected from the robots in E_X randomly. Once the current coordinator is failed, another one can be an alternative. Thus, there are no centralized components.

Remark 9. In our method, the sensing range is $N-1$ sequential collision states in terms of the abstraction, and the communication range is larger than the sensing range thanks to the wireless network. During the communication process, each robot only needs to communicate with some robots within the communication range. With the intermediate robots, a robot can also achieve the states of robots out of its communication range. Such information transmission is acceptable since the communication speed is much higher than the physical motion speed. Besides, a robot activates the communication process once it receives a message from other robots or it wants to move forward.

Now we can conclude the communication complexity of our method in terms of communication rounds. Here a communication round means that a robot sends a message with any size to the receiver. Suppose there are N robots in the system. First, during the detection of the circuits, a robot needs to communicate with the robots on its path. In the worst case, each robot needs to communicate with the rest $N-1$ robots, so the number of communication rounds is $N(N-1) = N^2 - N$. With a sequence of communication, a robot can obtain the risky walks of other robots. The number of

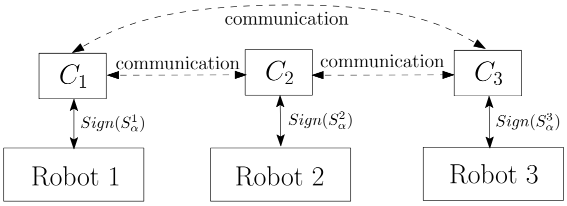


FIG. 7.10: An example of the control architecture of a multi-robot system under our approach. The dashed lines denote the communication among controllers.

communication rounds is at most N. Second, during the negotiation process, the largest number of communication rounds is N. Thus, in the worst case, the number of communication rounds is $(N^2 - N) + N + N = N^2 + N$. So the communication complexity is $O(N^2)$.

At the end, we give an example in Fig. 7.10 to show the system-level control structure of a multi-robot system under the proposed control algorithm. C_1, C_2, and C_3 are three local controllers, equipped with Algorithms 6 and 7, of three robots, and they only need to communicate with each other.

7.5 Simulation Cases

In this section, we give simulations of a system with 8 robots. Fig. 7.11 shows the LTS model of this system. The current state of a robot is marked with its index. Different colors represent different robots. Our goal is to guarantee that the robots can move to the private states $s_0^1, \ldots, s_0^8$ (the dashed private ones) successfully at the initial configuration $c_0 = (s_{cur}^i)_{i=1}^8 = (s_1, s_6, s_7, s_{10}, s_{11}, s_{12}, s_{13}, s_{14})$. Thus, once a robot arrives at its dashed private state, we no longer consider its motion. Our simulation is implemented with MATLAB R2017a on a desktop running Windows 10, and equipped with an Intel(R) Xeon(R) CPU E5-1650 v3 3.5GHz and 16 GB of RAM.

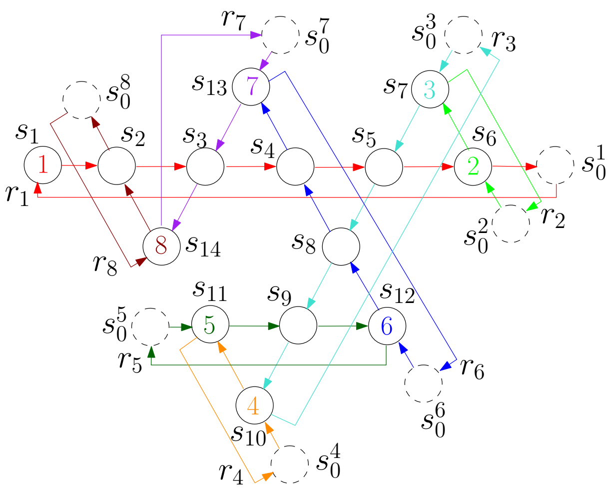


FIG. 7.11: A case study with 8 robots. Each dashed circle denotes a private state.

7.5.1 Simulation Without Higher-Order Deadlock Avoidance Algorithm

First, consider the evolution of the system without higher-order deadlock avoidance algorithm. Since all robots need to move into a crowded region with no private states, they have to negotiate. We apply Monte Carlo simulation method to do our simulation, i.e., each time the movable robots are randomly selected to move forward. Detailedly, we conduct 8 experiments with different simulation rounds: 10, 100, 500, 1000, 2000, 5000, 8000, and 10000. A round is an evolution of the system resulting in a deadlock or each robot to its dashed private state. A live round is an evolution of the system such that all robots can reach $s_0^1 - s_0^8$. To count the number of live rounds, we further repeat each experiment 100 times and then compute the average number of live rounds. Table 7.1 shows the number of simulation rounds and the corresponding average number of live rounds. In Fig. 7.12, we draw the numbers of total rounds and deadlock rounds, which are denoted by the star points. Then, we fit these points with a linear function, shown as the line in Fig. 7.12. Clearly, the slope of the line gives an experimental evaluation of the probability that an execution leads to a deadlock. In this experiment, the probability is around 0.3. This means the system is with low reliability without any deadlock avoidance algorithms.

TABLE 7.1: The Numbers of Simulation Rounds and Corresponding Average Live Rounds with Random Motion

# rounds	# live rounds	# rounds	# live rounds
10	3.19	2000	602.47
100	30.22	5000	1503.97
500	149.41	8000	2406.44
1000	296.9	10000	3017.65

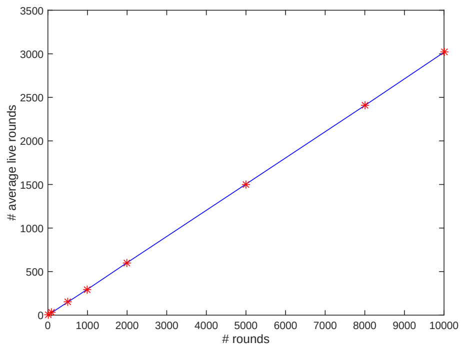


FIG. 7.12: The relation between the numbers of total rounds and average live rounds with random motion. Each experiment is repeated 100 times.

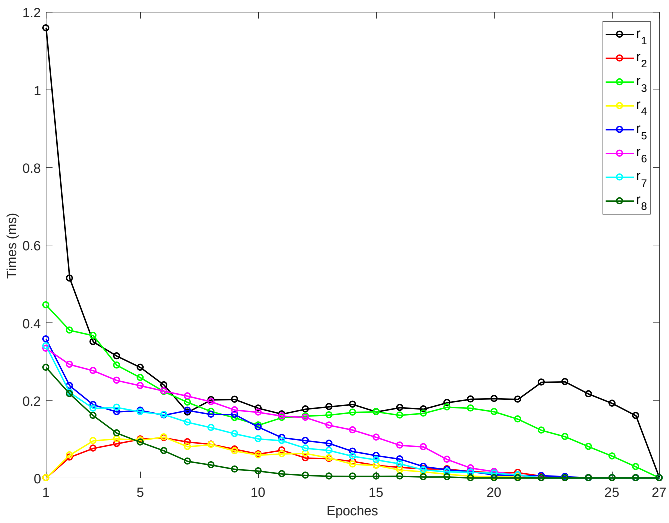
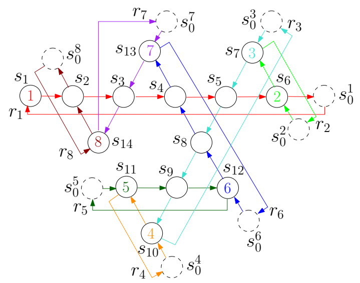


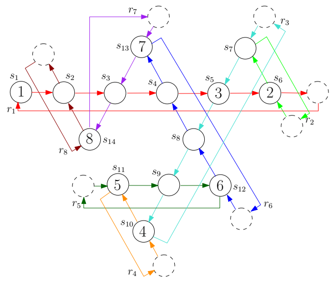
FIG. 7.13: Time estimation for higher-order prediction.

7.5.2 Simulation Under the Control of the Higher-Order Deadlock Avoidance Algorithm

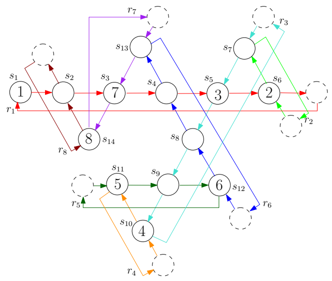
Next, we show the evolution of the system under the control of Algorithm 7. We run the system with the numbers of rounds shown in Table 7.1. The results show that none of them causes any deadlocks. At each round, all robots can reach $s_0^1 - s_0^8$ after total 27 steps of moves. To evaluate the efficiency of our approach, for each robot, we further



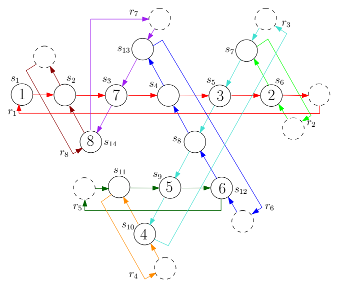
(a) Initial Configuration c_0.



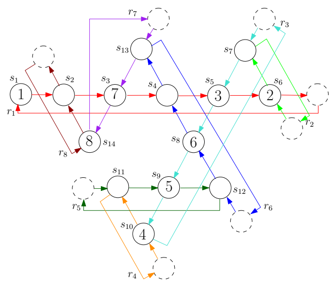
(b) Configuration c_1.



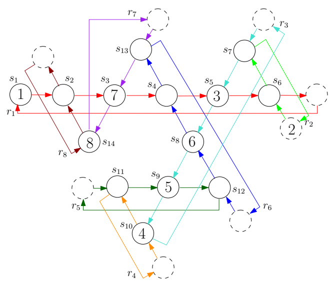
(c) Configuration c_2.



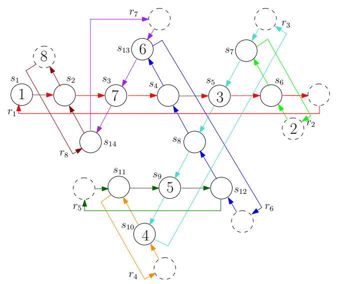
(d) Configuration c_3.



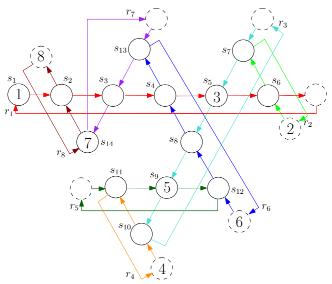
(e) Configuration c_4.



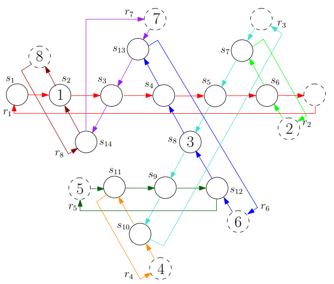
(f) Configuration c_5.



(g) Configuration c_6.



(h) Configuration c_7.



(i) Configuration c_8.

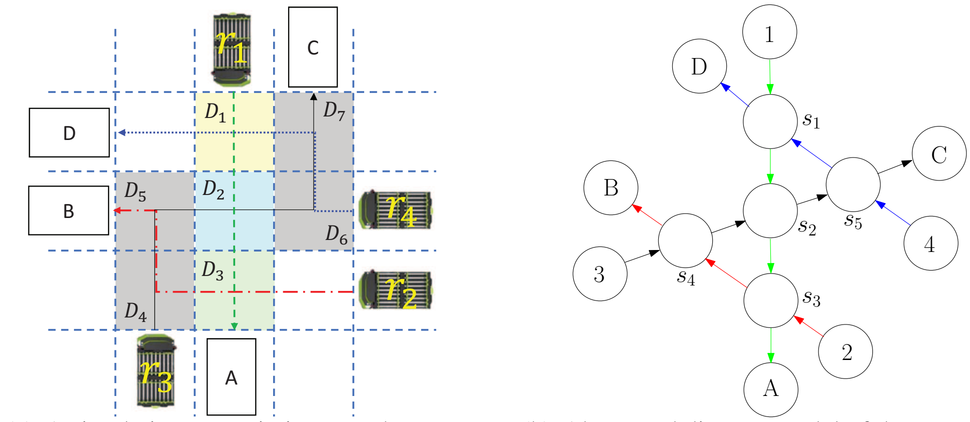
FIG. 7.14: Some snapshots during the evolution of the simulation system. (a) Initial Configuration c_0; (b) Configuration c_1 which is generated from c_0 by the move of r_3; (c) Configuration c_2 which is generated from c_1 by the move of r_7; (d) Configuration c_3 generated from c_2 by the move of r_5; (e) Configuration c_4 generated from c_3 by the move of r_6; (f); Configuration c_5 generated from c_4 after r_2 moves two steps; (g) Configuration c_6 generated from c_5 by the moves of $r_8r_6r_8r_6$; (h) Configuration c_7 generated from c_6 by the moves of r_4, r_7, r_6, and r_4; (i) Configuration c_8 generated by the moves of sequence $r_1r_5r_3r_7r_5$.

compute the time for higher-order deadlock detection at each step. We run the system 100 rounds and compute the average prediction time. The results are shown in Fig. 7.13. From Fig. 7.13, we can find that the prediction can be done in milliseconds. Note that at the initial configuration, r_1 predicts a higher-order deadlock with all other robots; while r_2 and r_4 do not need to perform the process of higher-order deadlock prediction since their next states are occupied by r_3 and r_5, respectively.

In the sequel, we show one evolution of the system from the simulation results. Via a set of negotiations, the sequence of the moves of robots is $r_3r_7r_5r_6r_2r_2r_8r_6r_8r_6r_4r_7r_6r_4r_1r_5r_3r_7r_5r_3r_1r_3r_1r_1r_3r_1r_1$. Fig. 7.14 shows some snapshots of this evolution. First, r_1, r_2, and r_4 cannot move forward since the move of r_1 can cause a fifth-order deadlock, while the succeeding states of r_2 and r_4 are occupied by other robots. Thus, the set of movable robots is $e_X = \{r_3, r_5, \ldots, r_8\}$, and r_3 obtains the right to move forward and moves to s_5. The generated configuration is $c_1 = (s_1,$ s_6, s_5, s_{10}, s_{11}, s_{12}, s_{13}, $s_{14})$, which is shown in Fig. 7.14(b). Clearly, r_4 is blocked by r_5 at c_1, so the robots that can move forward are $r_2, r_5, \ldots, r_8$. In the simulation, r_7 is the winner of the negotiation process. Thus, r_7 moves one step forward and the current configuration changes to $c_2 = (s_1,\ s_6,\ s_5,\ s_{10},\ s_{11},\ s_{12},\ s_3,\ s_{14})$, shown in Fig. 7.14(c). From this configuration, robots r_2, r_5, r_6, and r_8 are allowed to move, but finally only r_5 moves forward, resulting in the configuration c_3 shown in Fig. 7.14(d). Similarly, the next movable robot is r_6, and the generated configuration is shown in Fig. 7.14(e). The next two-step move of r_2 leads r_2 to its private state s_0^2, resulting in a configuration $c_5 = (s_1, s_0^2, s_5, s_{10}, s_9, s_8, s_3, s_{14})$, shown in Fig. 7.14(f). Next, we no longer consider the motion of r_2. With the next two successive moves of r_8r_6, the system arrives at configuration $c_6 = (s_1,\ s_0^2,\ s_5,\ s_{10},\ s_9,\ s_{13},\ s_3,\ s_0^8)$ shown in Fig. 7.14(g). From this configuration, after the move of r_4, r_7, r_6, and r_4 sequentially, r_4 and r_6 arrive at s_0^4 and s_0^6, respectively, as shown in Fig. 7.14(h). The next move sequence is $r_1r_5r_3r_7r_5$, and the system arrives at configuration c_8, shown in Fig. 7.14(i). At this configuration, all robots, except r_1 and r_3, arrive at their private states s_0^i, $i = 2, 4, 5, 6, 7, 8$, respectively. From c_8, r_1 and r_3 can finally move to their own private states.

7.5.3 Simulation on an Application Scenario in a Warehouse

In this subsection, we conduct a simulation on an application scenario in warehouse transportation. As shown in Fig. 7.15(a), in this scenario, four unmanned ground vehicles (UGVs) $r_1 - r_4$ are required to move to $A - D$, respectively. $D_1 - D_7$ are 7 collision regions in the warehouse. The lines in the collision regions are the paths for the vehicles. For example, r_3 is required to move through D_4, D_5, D_2, D_6, and D_7 to reach region C along the solid line. Based on our discretization method, the abstracted



(a) A simulation scenario in a warehouse. (b) Abstracted discrete model of the scenario.

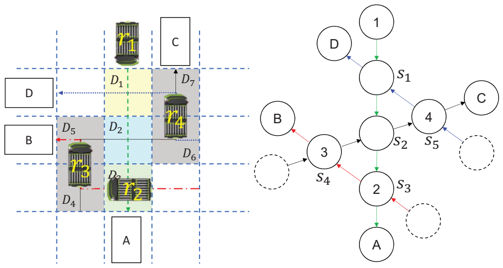
FIG. 7.15: An application scenario in a warehouse. (a) Four UGVs $r_1 - r_4$ are required to move to four targets along the predetermined paths, respectively. $D_1 - D_7$ are the collision region that two UGVs may collide. The lines in these regions are the paths of the UGVs. (b) The abstracted discrete state transition system of (a) based on our discretization method.

discrete model is given in Fig. 7.15(b), where D_4 and D_5 are abstracted to s_4, and D_6 and D_7 are abstracted to s_5.

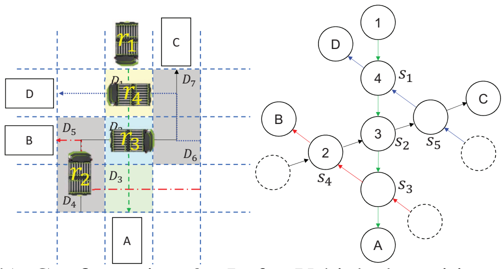
Our simulation results are shown in Fig. 7.16. Suppose r_2, r_3, and r_4 move into D_3, D_4, and D_6 first. This means they are at s_3, s_4, and s_5, respectively. Based on our method, r_1 cannot move into D_1, so r_1 is at s_1. Fig. 7.16(a) shows a snapshot of the four vehicles' positions and their related discrete states. Then r_2, r_3, and r_4 continue their motion and move into D_4, D_2, and D_1. During their motion in these regions, the discrete states are shown in Fig. 7.16(b). After passing through D_5 and D_1, r_2 and r_4 arrive at B and D successfully. Then r_1 can move into D_1. A snapshot of the configuration of the system and the related discrete states at this stage are shown in Fig. 7.16(c). Finally, r_1 and r_3 can arrive at A and C successfully, as shown in Fig. 7.16(d).

7.6 Discussion

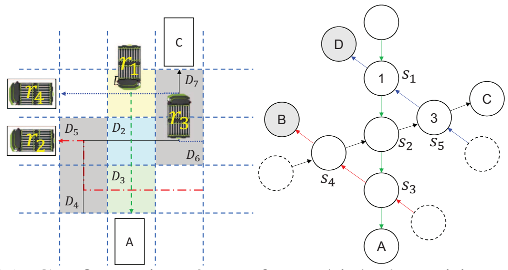
Many approaches have been proposed to deal with deadlocks in multi-robot systems. Some most related works are [130] and [162]. The comparison with [130] is given in Chapter 6. In the sequel, we would like to give a comparison with [162].



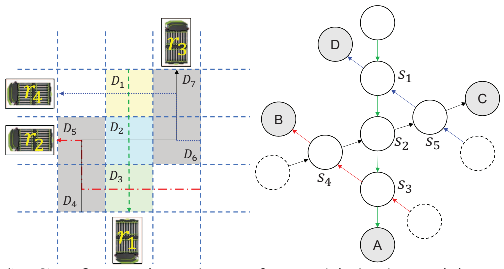
(a) Configuration 1: Left: Vehicles' positions; Right: Discrete states.



(b) Configuration 2: Left: Vehicles' positions; Right: Discrete states.

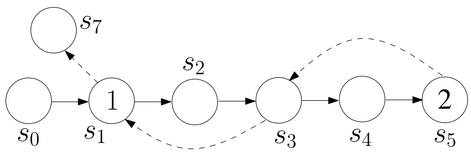


(c) Configuration 3: Left: Vehicles' positions; Right: Discrete states.

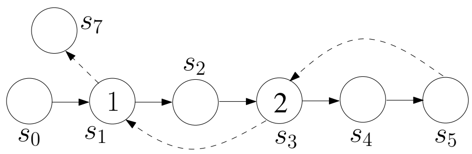


(d) Configuration 4: Left: Vehicles' positions; Right: Discrete states.

FIG. 7.16: Simulation results of the warehouse scenario. (a) – (d): Different configurations of the simulated system during its evolution based on the proposed algorithm.



(a) Configuration c_1



(b) Configuration c_2

FIG. 7.17: Example for the comparison of different methods. Solid arrows are the transitions of r_1 and dashed arrows are the transitions of r_2.

The work in [162] proposed a variant of Banker's algorithm to avoid deadlocks. The idea is that a robot can move forward only when it can arrive at a private stage that will not be occupied by other robots. The two methods are more conservative than ours. Indeed, given a multi-robot system, suppose the set of all reachable configurations is R, the sets of reachable configurations based on the methods of [162] and ours are R_1 and R_2, then we have $R_1 \subseteq R_2 \subseteq R$. For example, consider the configuration shown in Fig. 7.17(a). When the system is at this configuration, r_2 cannot move forward based on the method in [162] since it cannot move to its next private state; but r_2 can move forward based on our method, resulting in a reachable configuration c_2.

7.7 Conclusion

In this chapter, we further investigate higher-order deadlocks in multi-robot systems where each robot has a predetermined and closed path. Based on the discrete abstraction, we investigate the structural properties of higher-order deadlocks and conclude that the highest order of a higher-order deadlock formed by N robots is $N-3$. Then, a distributed algorithm is developed to avoid collisions and deadlocks. To perform its local algorithm in a distributed way, each robot needs to check the status of its collision states beforehand and communicate with others via a multi-hop communication path.

Chapter 8

Distributed Approach to Robust Control for Multi-Robot Systems

In Chapters 4, 6, and 7, we study motion control for multi-robot systems by assuming that all robots are reliable. However, robots with priori known levels of reliability may be used in applications to account for: 1) The cost in terms of unit price per robot type since higher reliability comes at a higher price, and 2) the cost in terms of robot wear in long term deployment due to the expensiveness of replacement (e.g., busses, trams, and subways). In this chapter, dividing robots into reliable and unreliable, we investigate robust control for multi-robot systems, which means that a failed robot has the least influence on the whole system. For the system studied in Chapter 4, robustness can be achieved by regarding the failed robots as static obstacles. However, for the system studied in Chapters 6 and 7, some robots may be blocked inevitably by the failed robots. Hence, robust control in such a system should minimize the number of robots whose motion is blocked by the failed ones. Based on the LTS models obtained in Chapter 5, in this chapter, to minimize the number of blocked robots, we propose a distributed approach to robust control, which contains two kinds of local algorithms: one for reliable robots and the other for unreliable ones.

8.1 Introduction

In practice, robots may fail during their motion due to different reasons, such as hardware wear, software failures, and cyber attacks. In case of robot failures, robust control, i.e., how to deal with robot failures and minimize their detrimental effect on the system, is important for robots. If robots can change their paths in motion, then as described in Chapter 4, robust control against robot failures regards the failed robots as obstacles during trajectory planning. However, for systems with fixed paths, robust control is significant but not easy to achieve. Hence, in this chapter, we focus on robust control in the system where each robot has a fixed path.

We assume that a system is configured with robots of different levels of reliability since the following factors: (1) Robots of higher reliability are more expensive. Sometimes, it is not cost-efficient to use robots of higher reliability. For example, for non-critical tasks like warehouse operations, it is more cost-efficient to repair the failed robots, rather than deploy higher-reliability robots; for dangerous environments like mining, we prefer cheap robots since we can replace the failed robots once they are broken and cannot be recovered. (2) For long-term robot deployment, hardware wear of robots determines the robot reliability. As time elapses, performance of robots can degrade gradually, and manufacturers oftentimes provide performance degradation information in the robots' technical manuals. We label robots of higher reliability as reliable ones, which are assumed to always work well, and those of lower reliability as unreliable ones, which may fail unexpectedly.

Here we describe the reliability of robots in a non-stochastic manner. A probabilistic analysis of robustness with respect to stochastic models of failures is left for future work. In this chapter, we assume that a classifier is available that can $a\ priori$ label robots as *reliable* and *unreliable* ones. Such a classifier might be provided by the robot manufacturer in terms of wear and/or robot models. By assuming collision and deadlock avoidance is always ensured based on Chapter 6 or 7, we propose two distributed algorithms for robust control: one is for reliable robots, while the other is for unreliable ones. Under the proposed algorithms, the failure of an unreliable robot blocks the minimum number of robots.

The main contributions of this study are:

- We investigate robust control in a multi-robot system. The control aims to minimize the number of stopped robots because of robot failures in multi-robot systems.
- We propose a distributed robust control approach, with which a robot only needs some local information to perform its motion via detecting its own path and communicating with its neighbors.

The chapter is organized as follows. Section 8.2 describes the problem statement. In Section 8.3, detailed algorithms for robust control are described. Simulation results are given in Section 8.4. Finally, discussion and conclusion are provided in Section 8.5.

8.2 Problem Statement

Based on the discrete model described in Chapter 5, we formulate the robust control problem studied in this chapter. First we introduce some notations. Given the set of robots $\{r_i : i \in \mathbb{I}_N\}$, the set of unreliable robots is $\{r_i : i \in \mathbb{U}_N\}$, where $\mathbb{U}_N \subseteq \mathbb{I}_N$. In the set of collision states S^i_α, the set of collision states that are passed by unreliable robots is denoted as ${}_uS^i_\alpha$, while the rest are denoted as ${}_rS^i_\alpha$. Therefore, ${}_uS^i_\alpha = \cup_{j \in \mathbb{U}_N \setminus \{i\}} S^j_\alpha \cap S^i_\alpha$, and ${}_rS^i_\alpha = S^i_\alpha \setminus {}_uS^i_\alpha$. The states in ${}_uS^i_\alpha$ and ${}_rS^i_\alpha$ are called unreliable and reliable collision states, respectively.

Definition 28 (Blocking). A robot r_i is said to block the motion of r_j if r_i stops at a state of S^j. We say r_j is blocked by r_i.

For example, as shown in Fig. 8.1, if r_1 stops at s_1. Since $s_1 \in S^6$, r_1 blocks the motion of r_6, and r_6 is blocked by r_1.

Definition 29 (Blocked Robots). Suppose an unreliable robot r_k fails at a state s. Let $B^1_{k,s} = \{r_i | s \in S^i\}$, $B^{l+1}_{k,s} = B^l_{k,s} \cup \{r_i \notin B^l_{k,s} | s^j_{cur} \in S^i, r_j \in B^l_{k,s}\}$, $l \geq 1$. Then, the set of blocked robots due to the failure of r_k, denoted as $B_{k,s}$, is $B_{k,s} = B^{l_0+1}_{k,s}$, where $B^{l_0+1}_{k,s} = B^{l_0}_{k,s}$.

In the above definition, the number of recursions cannot be larger than N, meaning that $l_0 < N$. Therefore, the recursive definition is well-defined. Note that the definition

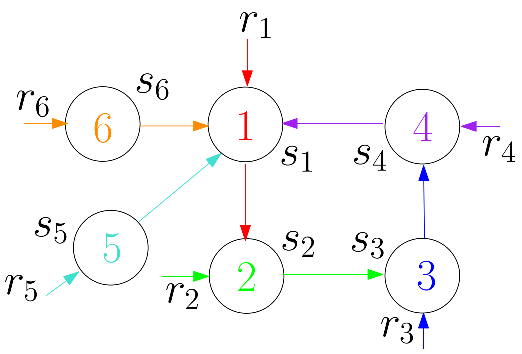


FIG. 8.1: An example illustrating robot blocking and blocked robots. r_1 is an unreliable robot and fails at s_1, and thus $r_2 - r_6$ are blocked.

of blocked robots is dependent on the position where the unreliable robot r_k fails. If r_k works well or fails at a private state, then the set of blocked robots is empty. When r_k fails at a collision state s, the robots in $B^1_{k,s}$ are blocked inevitably and are called directly blocked robots. Robots in $B^{\Delta}_{k,s} = B_{k,s} \setminus B^1_{k,s}$ are blocked indirectly by r_k.

For example, in Fig. 8.1, when r_1 fails at s_1, $B^1_{1,s_1} = \{r_4, r_5, r_6\}$. Since r_3 is blocked by r_4, $B^2_{1,s_1} = B^1_{1,s_1} \cup \{r_3\}$. r_2 is blocked by r_3, so $B^3_{1,s_1} = B^2_{1,s_1} \cup \{r_2\}$. There are no new robots that are blocked, so $B^4_{1,s_1} = B^3_{1,s_1}$. Hence, $B_{1,s_1} = B^4_{1,s_1}$ and $B^{\Delta}_{1,s_1} = \{r_2, r_3\}$.

Definition 30 (Robustness). A multi-robot system is robust if $\forall k \in \mathbb{U}_N$ and $\forall s \in S^k$, $B^{\Delta}_{k,s} = \emptyset$.

Note that if no robot fails in a system or a robot fails at a private state, then it always has $B^{\Delta}_{k,s} = \emptyset$, and hence the system is robust. In the sequel, our problem can be stated as follows:

Problem 4. Given a multi-robot system $\{\mathcal{T}_i\}_{i \in \mathbb{I}_N}$ with M unreliable robots $\{\mathcal{T}_j\}_{j \in \mathbb{U}_N}$, find an online and distributed control policy for the system such that the system is robust.

8.3 Robust Control

This section shows the development of two distributed robust control algorithms: one is for reliable robots and the other is for unreliable robots.

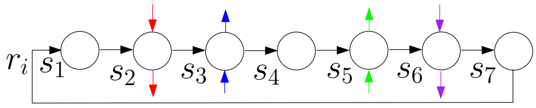


FIG. 8.2: An example to show critical states and critical pairs. In this LTS model of r_i, s_2, s_3, s_5, and s_6 are collision states.

8.3.1 Robust Control Algorithms

Definition 31 (Critical State). Let $\mathcal{T}_i$ be the LTS model of r_i. A state $x \in S^i$ is called a critical state of r_i if it satisfies: (1) $x \in S^i_\beta$ and $Pos_i(x) \in S^i_\alpha$; or (2) $x \in S^i_\beta$ and $Pre_i(x) \in S^i_\alpha$.

We call the states satisfying the first condition pre-critical states, denoted as $\mathscr{C}^i_1$, while call those satisfying the second condition post-critical states, denoted as $\mathscr{C}^i_2$. According to our assumptions in Chapter 5, $\mathscr{C}^i_1 \neq \emptyset$ and $\mathscr{C}^i_2 \neq \emptyset$ for any robot r_i. A pre-critical state is the last private state of a sequence of private states, while a post-critical state is the first private state of a sequence of private states. For example, as shown in Fig. 8.2, $\mathscr{C}^i_1 = \{s_1, s_4\}$ and $\mathscr{C}^i_2 = \{s_4, s_7\}$.

For any two states $x, y \in S^i$, let $x \prec_i y$ (or $y \;_i\!\succ x$) denote the relation that r_i can move from x to y within $|S^i|$ steps. The trace of $x \prec_i y$ is the sequence of states through which r_i passes from x to y. Since each robot in the system is performing persistent motion, $\forall x$, $y \in S^i$ and $x \neq y$, we have $x \prec_i y$ and $y \prec_i x$. Moreover, $x \prec_i y \prec_i z$ if and only if y is in the trace of $x \prec_i z$. For example, in Fig. 8.2, we have $s_3 \prec_i s_4 \prec_i s_5$. Even though we have $s_3 \prec_i s_5$ and $s_5 \prec_i s_4$, the notation $s_3 \prec_i s_5 \prec_i s_4$ is illegal since s_5 is not in the trace of $s_3 \prec_i s_4$. Indeed, the trace of $s_3 \prec_i s_4$ has only one state s_4.

Definition 32 (Critical Pair). Let $x \in \mathscr{C}^i_1$ and $y \in \mathscr{C}^i_2$. The ordered pair (x, y) is a critical pair if all states between x and y are collision states. For any collision state s between a critical pair (x, y), x is called the preceding critical state of s in S^i, denoted as $x <_i s$ (or $s \;_i\!> x$), and y is called the succeeding critical state of s in S^i, denoted as $s <_i y$ (or $y \;_i\!> s$).

The states between a critical pair form a maximal continuous subsequence of collision states (MCSS-CS). In the following, the MCSS-CS bordered by a critical pair (x, y) is denoted as $Z^i_{(x,y)} = \{s \mid x <_i s <_i y\}$. Clearly, $Z^i_{(x,y)} \subseteq S^i_\alpha$. For example, in Fig. 8.2, the critical pairs of r_i are (s_1, s_4) and (s_4, s_7), and $Z^i_{(s_1,s_4)} = \{s_2, s_3\}$ and $Z^i_{(s_4,s_7)} = \{s_5, s_6\}$. The preceding and succeeding critical states of s_2 are s_1 and s_4, respectively.

Proposition 5. $\forall x \in \mathscr{C}^i_1$, $\exists! y \in \mathscr{C}^i_2$, $\ni (x, y)$ is a critical pair in S^i, and vice versa.

Proof. Based on Definition 31, $\forall x \in \mathscr{C}^i_1$, $Pos_i(x) \in S^i_\alpha$; based on Definition 32, only the first private state starting from $Pos_i(x)$ can form a critical pair with x. Similarly, $\forall y \in \mathscr{C}^i_2$, $Pre_i(y)$ is a collision state; searching back from it, the first private state constructs a critical pair with y. Clearly, in either case, the first found state is unique. □

This proposition states that any MCSS-CS is bordered by a unique critical pair.

Proposition 6. $\forall s \in S^i_\alpha$, (1) $\exists! x \in \mathscr{C}^i_1$, $\ni x <_i s$. (2) $\exists! y \in \mathscr{C}^i_2$, $\ni s <_i y$.

Proof. On one hand, there exists a state x, $x \in S^i_\beta$, such that $\forall z$, $x \prec_i z \prec_i s$, $z \in S^i_\alpha$. Indeed, x can be found as follows. Let $ps = Pre_i(s)$, check ps and set $ps = Pre_i(ps)$ recursively until $ps \in S^i_\beta$. Thus, the returned ps is the first preceding private state of s. Clearly, it is the required preceding critical state of s and is unique. On the other hand, we can similarly find the first succeeding private state of s. Let $ss = Pos_i(s)$ and iterate ss with $ss = Pos_i(ss)$ until $ss \in S^i_\beta$. The returned state is the succeeding critical state of s. The first one is unique. □

In the proof of Proposition 6, the states traversed by ps and ss, except the two end states, constitute an MCSS-CS. Thus, this proposition states that any collision state s only belongs to one MCSS-CS.

Proposition 7. (1) $S^i_\alpha = \cup_{x \in \mathscr{C}^i_1} Z^i_{(x,y)}$; (2) $Z^i_{(x_1,y_1)} \cap Z^i_{(x_2,y_2)} = \emptyset$ for $x_1 \neq x_2$.

Proof. (1) On one hand, based on the definition of critical pair, $\forall x \in \mathscr{C}^i_1$, $Z^i_{(x,y)} \subseteq S^i_\alpha$. Thus, $\cup_{x \in \mathscr{C}^i_1} Z^i_{(x,y)} \subseteq S^i_\alpha$. On the other hand, based on Proposition 6, each collision state belongs to an MCSS-CS. Since the states of all MCSSs-CS are $\cup_{x \in \mathscr{C}^i_1} Z^i_{(x,y)}$, $S^i_\alpha \subseteq \cup_{x \in \mathscr{C}^i_1} Z^i_{(x,y)}$. Hence, $S^i_\alpha = \cup_{x \in \mathscr{C}^i_1} Z^i_{(x,y)}$. (2) If $\exists s \in Z^i_{(x_1,y_1)} \cap Z^i_{(x_2,y_2)}$, s has two preceding critical states. This contradicts Proposition 6. □

The above proposition clarifies that for any robot r_i, all of its MCSS-CS form a partition of S_α^i.

Based on the above description, we can now focus on each MCSS-CS. For any critical pair (x, y) in S^i, let $US_x^i = Z_{(x,y)}^i \cap {}_uS_\alpha^i$. If r_i's current state $s_{cur}^i \in Z_{(x,y)}^i$, the set $Z_{(x,y),s_{cur}^i}^i = \{s \mid s_{cur}^i \prec_i s \prec_i y\}$ is called a block-risk set of r_i. Indeed, $Z_{(x,y),s_{cur}^i}^i$ is the remaining set of collision states in the current MCSS-CS that r_i needs to traverse.

Theorem 9. If a multi-robot system satisfies that at any time, there are no robots whose block-risk sets contain unreliable robots, then the system is robust.

Proof. Suppose that robot r_k is an arbitrary unreliable robot and fails at s. If $s \in S_\beta^k$, there are no robots that are blocked. If $s \in S_\alpha^k$, the set of directly blocked robots is $S_{k,s}^1$. $\forall r_i \in S_{k,s}^1$, suppose (x, y) is the critical pair of s in S^i. Clearly, r_i's current state s_{cur}^i cannot be in $Z_{(x,y)}^i$; otherwise, r_i's block-risk set contains an unreliable robot r_k. This means that r_i can only arrive at x eventually. Therefore, r_i cannot block other robots due to the failure of r_k. Hence, $B_{k,s}^\Delta = \emptyset$. Thus, the system is robust. □

Based on Theorem 9, the policy of robust control can be defined as follows: For any robot r_i and any critical pair (x, y) in S^i, (1) if r_i enters $Z_{(x,y)}^i$ first, all unreliable robots cannot move to r_i's block-risk set and (2) if there is an unreliable robot in $Z_{(x,y)}^i$, then r_i cannot move into $Z_{(x,y)}^i$ unless all unreliable robots move away. This policy can be implemented as follows. Let $Flag$ be a $|{}_uS_\alpha| \times N$-dimensional Boolean matrix, denoting whether an unreliable collision state is in some robot's block-risk set. $Flag(s, i) = 1$ means that the unreliable collision state s is in the block-risk set of r_i. The signals in $Flag$ only affect the motion of unreliable robots. Let $Sign_u$ be a $|{}_uS_\alpha|$-dimensional Boolean vector, denoting the status of unreliable collision states. $Sign_u(s) = 1$ means that the unreliable collision state s is occupied by an unreliable robot.

Now, we present the robust control framework. In the following description, we assume that deadlock avoidance is presumptively ensured since the deadlock avoidance strategy should be performed first.

Consider reliable robots. If a reliable robot r_i is at x, $x \in \mathscr{C}_1^i$, and is about to move into $Z_{(x,y)}^i$, it must guarantee that there are no unreliable robots at the states in $Z_{(x,y)}^i$.

Thus, r_i first checks the value of $Sign_u$. If $\exists s \in US_x^i$, $\ni Sign_u(s) = 1$, r_i stops at x. Otherwise, the negotiation process $NEG(E_X)$ is executed immediately. If it gets the right to move, r_i moves into $Z_{(x,y)}^i$ and sets $Flag(s,i) = 1$ for all $s \in US_x^i$. This setting will prevent unreliable robots from moving into r_i's block-risk set. Once r_i leaves a state $z \in US_x^i$, $Flag(z,i) = 0$.

Consider unreliable robots. The motion of an unreliable robot r_k should guarantee two things: (a) no other unreliable robots can be in r_k's block-risk set and (b) r_k cannot move into another robot's block-risk set. The former means that r_k cannot be blocked in a collision state by other unreliable robots, while the latter means that r_k cannot block other robots if $r_{k'}$ fails at s. First, suppose r_k is at x, $\forall x \in \mathscr{C}_1^k$. To move forward, r_k first checks the value of $Flag$. If (i) $\exists s \in US_x^k$, $j \in \mathbb{U}_N \setminus \{k\}$, $\ni Flag(s,j) = 1$ (including the case $Sign_u(s) = 1$), or (ii) $\exists i \in \mathbb{I}_N \setminus \{k\}$, $\ni Flag(Pos_k(x), i) = 1$, r_k stops at x. Note that the negation of the first condition guarantees that no other unreliable robots can be in $Z_{(x,y)}^k$, and that of the second one guarantees that r_k cannot move into other robots' block-risk sets. Therefore, r_k can move to $Pos_k(x)$. Then, r_i executes $NEG(enable)$ immediately. If it gets the right to move, r_i moves one step forward, causing $Flag(z,k) = 1$ for all $z \in US_x^k$ and $Sign_u(Pos_k(x)) = 1$. Thus, no other unreliable robots can move into r_k's block-risk set. Hence, (a) is always guaranteed during r_k's motion in $Z_{(x,y)}^k$. Second, suppose r_k is at a collision state s, $s \in Z_{(x,y)}^k$. As described before, (a) is always guaranteed once r_k has moved into $Z_{(x,y)}^k$, so r_k only needs to guarantee (b) during the motion in $Z_{(x,y)}^k$. This can be implemented by checking the values of $Flag$ with respect to its succeeding state. If $\exists i \in \mathbb{I}_N \setminus \{k\}$, $\ni$ $Flag(Pos_k(s), i) = 1$, r_k stops at its current state s. Once r_k leaves s, $Sign_u(s) = 0$ and $Flag(s,k) = 0$ if $s \in US_x^k$. Moreover, if it fails at s, then $\forall z \in Z_{(x,y),s}^k \cap {}_uS_\alpha^k$, $Flag(z,k) = 0$.

The details are given in Algorithms 8 and 9. In Algorithm 8, Lines 2–7 treat the situation in which the succeeding state is a private state. In such a situation, r_i can always move forward and release the corresponding signals (Lines 4–6). Lines 8–32 execute the situation in which no deadlocks are detected. It contains two sub-cases. Lines 9–21 are executed when r_i is at a preceding critical state. Once a robot arrives at a collision state, unreliable robots cannot move to its block-risk set. This is performed by Lines 22−32.

Algorithm 9 is almost the same as Algorithm 8 except the following special aspects. First, when it is at a preceding critical state (performing Lines 10−22), r_k should check not only whether unreliable robots exist, but also check the values of $Flag$ with respect to its succeeding state, i.e., Lines 12 and 13. Second, if r_k is currently at a collision state (performing Lines 23−36), it needs to check the values of $Flag$ corresponding to its succeeding state, i.e., Lines 24 and 25. Third, if r_k fails at a collision state, it releases its signals in $Flag$, i.e., Lines 39−40.

8.3.2 Effectiveness Analysis

In this part, the effectiveness of the algorithms is given.

Lemma 5. For any state $s \in {}_uS_\alpha$, $\sum_{i\in\mathbb{U}_N} Flag(s,i) \leq 1$.

Proof. Suppose r_k is an unreliable robot. $\forall x \in \mathscr{C}_1^k$, r_k can move to $Pos_k(x)$ only when $Flag(i,s) = 0$ for all $i \in \mathbb{U}_N$ and $s \in US_x^k$ based on Lines 12 − 16 in Algorithm 9. Once r_k moves to $Z^k_{(x,y)}$, $\forall s \in US_x^k$, $Flag(s,k) = 1$. Thus, based on Line 12 in Algorithm 9, other unreliable robots $r_{k'}$ cannot move into their own $Z^{k'}_{(x',y')}$ containing the states in US_x^k. Thus, if $Flag(s,k) = 1$, then $\forall i \in \mathbb{U}_N \setminus \{k\}$, $Flag(s,i) = 0$. Note that there may be that $\forall i \in \mathbb{U}_N$, $Flag(s,i) = 0$, such that all unreliable robots are at their private states. Hence, $\sum_{i\in\mathbb{U}_N} Flag(s,i) \leq 1$. Since ${}_uS_\alpha = \cup_{k\in\mathbb{U}_N} \cup_{x\in\mathscr{C}_1^k} US_x^k$, applying this result to each US_x^k, $\forall k \in \mathbb{U}_N$ and $\forall x \in \mathscr{C}_1^k$, we complete the proof. □

This lemma states that for any unreliable collision state s, there exists at most one unreliable robot, say r_k, such that $Flag(s,k) = 1$. This means that s cannot be in two unreliable robots' block-risk sets simultaneously.

Lemma 6. If a reliable robot can move to a collision state, then it can eventually move to a post-critical state.

Proof. Consider a reliable robot r_i. For any collision state $s \in S_\alpha^i$, suppose its critical pair in S^i is (x,y), i.e., $x <_i s <_i y$. We need to prove that during r_i's motion in $Z^i_{(x,y)}$,

Algorithm 8: Robust control for a reliable robot r_i.

Input: $\mathcal{T}_i = \langle S^i, \Sigma_i, \rightarrow_i \rangle$, its current state s_{cur}, signals $Sign$, $Sign_u$, and $Flag$.

Initialization: $s_{next} := Pos_i(s_{cur})$ and determine the negotiation region X;
if $s_{next} \in S^i_\beta$ **then**
    execute the transition $s_{cur} \xrightarrow{move}_i s_{next}$;
    if $s_{cur} \in S^i_\alpha$ **then**
        $Sign(s_{cur}) := 0$;
        $Flag(s_{cur}, k) = 0$ if $s_{cur} \in {}_uS^k_\alpha$;
    $s_{cur} := s_{next}$, $s_{next} := Pos_i(s_{cur})$;
else if $Sign(s_{next}) = 0 \wedge$ no deadlocks **then**
    /* There are no collisions or deadlocks. */
    if $s_{cur} \in \mathscr{C}^i_1$ **then**
        /* r_i is at a pre-critical state. */
        $V := US^i_{s_{cur}}$;
        if $\exists s \in V, \ni Sign_u(s) = 1$ **then**
            stop its motion for a proper duration;
        else
            Add r_i to E_X;
            if $NEG(E_X) = r_i$ **then**
                execute the transition $s_{cur} \xrightarrow{move}_i s_{next}$;
                $\forall z \in V$, $Flag(z, i) := 1$;
                $s_{cur} := s_{next}$, $s_{next} := Pos_i(s_{cur})$;
                $Sign(s_{cur}) := 1$, $E_X := \emptyset$;
            else
                stop its motion for a proper duration;
    else
        /* r_i is at a collision state. */
        Add r_i to E_X;
        if $NEG(E_X) = r_i$ **then**
            execute the transition $s_{cur} \xrightarrow{move}_i s_{next}$;
            $Sign(s_{cur}) := 0$, $Sign(s_{next}) := 1$;
            if $s_{cur} \in {}_uS^i_\alpha$ **then**
                $Flag(s_{cur}, i) := 0$
            $s_{cur} := s_{next}$, $s_{next} := Pos_i(s_{cur})$;
            $E_X = \emptyset$;
        else
            stop its motion for a proper duration;
else
    /* $Sign(s_{next}) = 1$ or there is a deadlock */
    stop its motion for a proper duration;

Algorithm 9: Robust control for an unreliable robot r_k.

Input: $\mathcal{T}_k = \langle S^k, \Sigma_k, \rightarrow_k \rangle$, its current state s_{cur}, signals $Sign$, $Sign_u$, and $Flag$.

Initialization: $s_{next} := Pos_k(s_{cur})$ and determine negotiation region X;
if r_k works well **then**
  if $s_{next} \in S\beta^k$ **then**
    execute the transition $s_{cur} \xrightarrow{move}_k s_{next}$;
    if $s_{cur} \in S^k_\alpha$ **then**
      $Sign(s_{cur}) := 0$, $Sign_u(s_{cur}) := 0$;
      $Flag(s_{cur}, k) = 0$ if $s_{cur} \in {}_uS^k_\alpha$;
    $s_{cur} = s_{next}$, $s_{next} = Pos_k(s_{cur})$;
  else if $Sign(s_{next}) = 0 \wedge$ no deadlocks **then**
    if $s_{cur} \in \mathscr{C}^k_1$ **then**
      /* r_k is at a pre-critical state. */
      $V := US^k_{s_{cur}}$;
      if $(\exists s \in V, j \in \mathbb{U}_N \setminus \{k\}, \ni Flag(s, j) = 1) \mid\mid$ $(\exists i \in \mathbb{I}_N \setminus \{k\}, \ni Flag(s_{next}, i) = 1)$ **then**
        stop the motion for a proper duration;
      else
        Add r_k to E_X;
        if $NEG(E_X) = r_k$ **then**
          execute transition $s_{cur} \xrightarrow{move}_k s_{next}$;
          $\forall z \in V$, $Flag(z, k) := 1$;
          $s_{cur} := s_{next}$, $s_{next} := Pos_k(s_{cur})$;
          $Sign(s_{cur}) := 1$, $Sign_u(s_{cur}) := 1$, $E_X := \emptyset$;
        else
          stop its motion for a proper duration;
    else
      /* r_k is at a collision state. */
      if $\exists j \in \mathbb{I}_N \setminus \{k\}, \ni Flag(s_{next}, j) = 1$ **then**
        stop the motion for a proper duration;
      else
        Add r_k to E_X;
        if $NEG(E_X) = r_k$ **then**
          execute transition $s_{cur} \xrightarrow{move}_k s_{next}$;
          $Sign(s_{cur}) := 0$, $Sign_u(s_{cur}) := 0$;
          $Sign(s_{next}) := 1$, $Sign_u(s_{next}) := 1$;
          if $s_{cur} \in {}_uS^k_\alpha$ **then**
            $Flag(s_{cur}, k) = 0$
          $s_{cur} := s_{next}$, $s_{next} := Pos_k(s_{cur})$; $E_X := \emptyset$;
        else
          stop its motion for a proper duration;
  else
    stop its motion for a proper duration;
else if r_k fails at a collision state s_{cur} **then**
  $\forall z \in Z^k_{(x,y),s^k_{cur}} \cap {}_uS^k_\alpha$, $Flag(z, k) := 0$ when $s_{cur} \in S^k_\alpha$;

no unreliable robots can move to $Z^i_{(x,y),s^i_{cur}}$. First, suppose the reliable robot r_i is at x, $x \in \mathscr{C}^i_1$. If r_i can move to $Pos_i(x)$, based on Lines 11 and 15 in Algorithm 8, no unreliable robots can be at the states in US^i_x. Once r_i arrives at $Pos_i(x)$, $Flag(s,i) = 1$, $\forall s \in US^i_x$. Based on Lines 12 and 24 in Algorithm 9, no unreliable robots can move to the states in US^i_x. During r_i's motion in $Z^i_{(x,y)}$, there are no unreliable robots at the states in $US^i_x \cap Z^i_{(x,y),s^i_{cur}}$. Thus, r_i can eventually move to y under the control of deadlock avoidance strategy. Second, consider the general case. For any collision state s of r_i, there exist $x_0 \in \mathscr{C}^i_1$ and $y_0 \in \mathscr{C}^i_2$ such that $x_0 <_i s <_i y_0$. Thus, if it can move to s, r_i must first move to $Pos_i(x_0)$. By applying the previous result, r_i can eventually move to y_0. $\square$

Lemma 7. If an unreliable robot can move to a collision state, then it can either move to a post-critical state or fail.

Proof. Suppose r_k is an arbitrary unreliable robot. For any collision state s, the critical pair in S^k is (x,y), i.e., $x <_k s <_k y$. Thus, we need to prove that (1) $Flag$ cannot prevent the motion of r_k and (2) no other unreliable robots can move to $Z^k_{(x,y),s^k_{cur}}$. We first consider that r_k is at x, $x \in \mathscr{C}^k_1$, and can move to $Pos_k(x)$. On one hand, based on Lines $12 - 22$ in Algorithm 9, we have $Flag(Pos_k(x), i) = 0$ for all $i \in \mathbb{I}_N$ and $Flag(s,j) = 0$ for all $s \in US^k_x$, $j \in \mathbb{U}_N$. Once r_k arrives at $Pos_k(x)$, $Flag(s,k) = 1$ for all $s \in US^k_x$. Based on the proof of Lemma 5, for any state s in r_k's block-risk set, $Flag(s,j) = 0$ where $i \in \mathbb{U}_N \setminus \{k\}$, and $Flag(s,i)$, $i \in \mathbb{I}_N \setminus \mathbb{U}_N$, will eventually be 0 based on Lemma 6. Thus, $Flag$ cannot block r_k's motion to the end. On the other hand, based on Line 12 in Algorithm 9, with respect to other unreliable robots, during r_k's motion in $Z^k_{(x,y)}$, $Flag(s,k)$, $s \in US^k_x$, prevents other unreliable robots from moving into r_k's block-risk set. Thus, r_k can eventually move to y under the control of the deadlock avoidance strategy if it does not fail. Second, consider the general case. For any collision state s of r_k, there exist $x_0 \in \mathscr{C}^k_1$ and $y_0 \in \mathscr{C}^k_2$ such that $x_0 <_k s <_k y_0$. Thus, if it can move to s, r_k must first move to $Pos_k(x_0)$. By applying the previous result, r_k can eventually move to y_0 if it does not fail. $\square$

Theorem 10. The system is robust under the control of Algorithms 8 and 9.

Proof. Note that if it fails at a private state, a robot cannot affect others. Thus, we only need to consider the case in which an unreliable robot fails at a collision state. Based on Lines 39 and 40 in Algorithm 9, an unreliable robot r_k will reset its corresponding signals in $Flag$ to 0 when it fails at a collision state. Thus, based on Lemmas 6 and 7, $Flag$ cannot affect the motion of each robot eventually. Based on Lines 11 and 12 in Algorithm 8 and Lines 12 and 13 in Algorithm 9, when r_k fails at a collision state, all directly blocked robots stop at their own preceding critical states of the failure location. Thus, they cannot block other robots, i.e., the set of indirectly blocked robots is empty. Hence, the system is robust. □

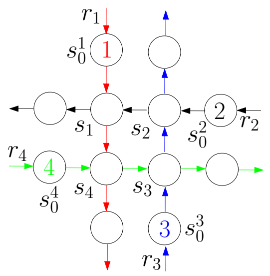
8.3.3 Distributivity and Complexity Analysis

The control of multi-robot systems admits three types of architectures: centralized, decentralized, and distributed. For centralized control, the whole system has only one global controller; for decentralized and distributed control, each subsystem has a local controller, but for distributed control, local controllers have communication. In this subsection, we analyze the distributed nature of the proposed control policies.

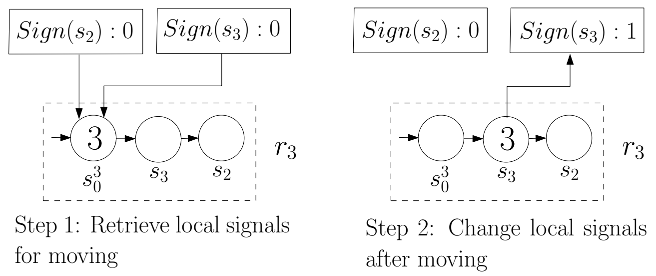
According to the algorithms, to execute the related algorithm, each robot may need to (1) retrieve the status of some collision states on its path and (2) communicate with its neighboring robots.

On one hand, by checking its local signal variables $Sign$, $Sign_u$, and $Flag$, r_i can retrieve the status of the collision states on its path. Indeed, during the implementation, the elements of these variables are divided into a set of separated local signals and stored in robots. Each time a robot only changes some of the local signals. By checking its owe path, r_i can retrieve the values of $Sign(s)$ and $Sign_u(s)$ for $s \in S_\alpha^i$, and $Flag(s, i)$ for $s \in_u S_\alpha^i$. By communicating with its neighbors r_k, r_i can further retrieve the values of $Flag(s, k)$ for $s \in_u S_\alpha^i$ and $s \in S^k$.

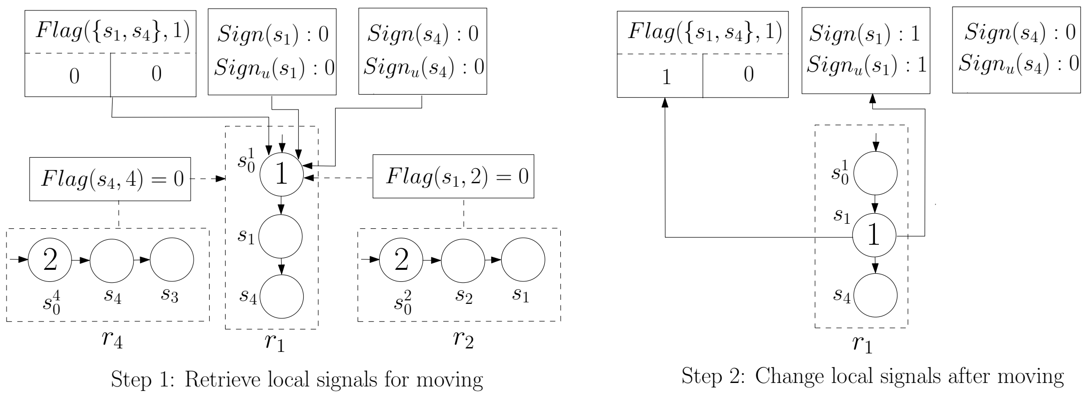
For example, consider the system shown in Fig. 8.3(a). Suppose there are four robots traversing this crossing and robot r_1 is an unreliable robot. Suppose the four robots are currently at $s_0^1 - s_0^4$. Consider the motion of r_1 and r_3 at the current configuration. Fig. 8.3(b) shows the case of r_3's motion. Since it is a reliable robot and no



(a) $r_1 - r_4$ are at $s_0^1 - s_0^4$ and r_1 is unreliable.



(b) Retrieval and change of local signals by r_3.



(c) Retrieval and change of local signals by r_1.

FIG. 8.3: An example of local signal retrieval and maintenance for robust control. The solid arrows denote the direct monitoring of local signals and the dashed arrows denote the communication among robots to retrieve the related signals.

unreliable robot passes through its path, r_3 only needs to check the status of its own collision states, i.e., the values of $Sign(s_2)$ and $Sign(s_3)$ when it is at s_0^3. Once r_3 moves to s_3, it only changes the value of $Sign(s_3)$. Fig. 8.3(c) shows the case of r_1's motion. Since r_1 is unreliable, r_1 needs to retrieve the signals $Sign(s_1)$, $Sign_u(s_1)$, $Flag(s_1, 1)$, $Flag(s_1, 2)$, and $Sign(s_4)$, $Sign_u(s_4)$, $Flag(s_4, 1)$, $Flag(s_4, 4)$ when it is at s_0^1. r_1 can retrieve the values of $Sign(s_1)$, $Sign_u(s_1)$, $Sign(s_4)$, $Sign_u(s_4)$, and $Flag(\{s_1, s_4\}, 1)$ directly by monitoring its path. By communicating with r_2 and r_4, r_1 can know the values of $Flag(s_1, 2)$ and $Flag(s_4, 4)$. Once it moves to s_1, r_1 then changes the values of $Sign(s_1)$, $Sign_u(s_1)$, $Flag(s_1, 1)$, and $Flag(s_4, 1)$.

In conclusion, $Sign$, $Sign_u$, and $Flag$ are collections of local signals, rather than global signals; a robot can retrieve their values by either checking its own paths directly or communicating with other robots. Besides, a local signal is maintained by an individual robot each time.

Lemma 8.1. *Under the control of Algorithms 8 and 9, the information about local signals needed by a robot is minimal.*

Remark 8.2. Here, the information is counted in terms of the number of communication messages. Each message contains the value of only one local signal. This means that if a package contains the values of two or more variables, it should be regarded as two or more communication messages.

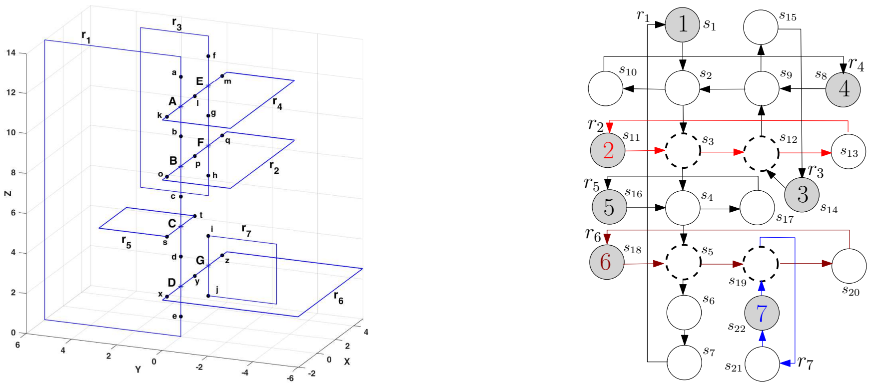
Proof. Based on the definition of robustness, to avoid blocking other robots, a robot cannot move into an MCSS-CS containing unreliable robots and an unreliable robot cannot move into others' block-risk sets. Thus, for any robust control scheme, a robot should check at least the status of the MCSS-CS states that it needs to move into; an unreliable robot should first check its succeeding state to determine whether its movement will lead it to the block-risk sets of other robots. Thus, such information is the minimal amount needed by any robust control.

Line 11 in Algorithm 8 is used for robustness checking. Only when it is at a preceding critical state x does r_i need the values of $Sign_u(s), \forall s \in US_x^i$. In Algorithm 9, Lines 12 and 24 indicate the procedures for robustness check. Line 12 checks the values of $Flag(s, j), \forall s \in US_x^k$ and $\forall j \in \mathbb{U}_N$, when r_k is at a preceding critical state x, while Line 24 checks the values of $Flag(s_{next}, j), \forall j \in \mathbb{I}_N \setminus \{k\}$. The former is to check the status of the states in MCSS-CS, while the latter is to check the status of the succeeding state. Clearly, in both algorithms, the needed information is the minimal information for robustness. □

Based on above discussion, we can conclude that:

Corollary 1. The motion control of the system under the proposed algorithms is distributed.

To this end, we provide the complexity analysis of the proposed approach. Based on the algorithms, a robot needs to perform three tasks to determine whether it can move forward. The first one is to propagate communication among robots for deadlock avoidance. The worst case is that the propagation is executed among all robots. Therefore,



(a) Continuous path network. (b) Discrete states.

FIG. 8.4: The system for our simulation. (a) $A - G$: intersections, and $a - z$: safe boundaries of intersections. (b) Abstracted discrete states.

the complexity is $O(N)$. The second one is to check the status of the states in its block-risk set, if any. In the worst case, there exists only one private state, while the others are collision states, resulting in a complexity of $O(|S^i|)$. The last one is to negotiate with others, if needed, to determine which one can eventually move. The worst case is that all robots in the system are trying to move to a same region; so the complexity of this case is $O(N)$. Hence, for robot r_i, the complexity is $O(|S^i| + N)$. Let $SN = \max_{i \in \mathbb{I}_N} |S^i|$. Since the robots in the system are moving in a distributed way, the final complexity of our control method is $O(SN + N)$.

8.4 Simulation Cases

In this section, we implement the algorithms in MATLAB. Some simulations for a system with seven robots $r_1, r_2, \ldots, r_7$ are demonstrated. The closed paths are shown in Fig. 8.4(a). $A, B, \ldots, G$ are seven intersections with coordinates $A(0, 0, 11.5)$, $B(0, 0, 8.5)$, $C(0, 0, 5.5)$, $D(0, 0, 2.5)$, $E(3, 0, 11.5)$, $F(3, 0, 8.5)$, and $G(3, 0, 2.5)$. Suppose a safe radius of $\rho = 1.5$ units for each robot. Thus, the safe boundaries of these intersections for the robots are given in lowercase letters. For example, for r_1, the safe boundaries of A are a and b, while the safe boundaries of A for r_4 are k and l. Therefore, the path segment pair (ab, kl) is a collision region of r_1 and r_4, and is abstracted as a collision state s_2, as shown in Fig. 8.4(b); so are other intersections. The discrete state space of the system is shown in Fig. 8.4(b). Moreover, we assume r_2, r_6, and r_7 are unreliable robots. Suppose the permutation of the configuration is (s^1, s^2, s^3, s^4, s^5, s^6, s^7), where s^i is the state of robot r_i, $i \in \mathbb{I}_7$. The initial configuration $c_0 = (s_1, s_{11}, s_{14}, s_8, s_{16}, s_{18}, s_{22})$. We consider the situation in which r_2 fails at s_3. Our experiments are carried out in two stages. The first one is to simulate the system only with collision and deadlock avoidance strategy, while the second one is to perform the simulation with the proposed robustness algorithms.

In our simulation, since the seven robots are moving in a small region, we assume that all of them are always connected transparently via communication and the transitions are enabled synchronously for convenience. However, strictly based on our approach, at most $\{r_1 - r_4\}$ need negotiation, so do $\{r_1, r_5\}$ and $\{r_1, r_6, r_7\}$.

8.4.1 Robot Motion without Robustness Algorithms

First, the system is controlled only by the collision and deadlock avoidance strategy. Because of the concurrency, there are many evolution traces of the system. Fig. 8.5 shows eight snapshots of one evolution trace of the system, where the filled states denote the current states of the robots.

Robot r_2 fails at s_3. First, all robots are able to move forward. Suppose after a round of negotiations, r_1 is allowed to move forward. Thus, r_1 moves one step forward. After the movement of r_1, the new movable robots are $r_1, r_2, \ldots, r_7$. Suppose r_2 moves one step forward at this time. Next, we assume that $r_3, r_4, \ldots, r_7$ get the right to move sequentially. Thus, the system reaches c_1, as shown in Fig. 8.5(a). At present, r_2 fails at s_3. Hence, the current movable robots are $r_3, r_4, \ldots, r_7$. Supposing that $NEG(\{r_3, r_4, \ldots, r_7\}) = r_3$, r_3 moves one step forward, which causes r_4 to be blocked. Moreover, in the next three rounds of negotiations, we assume that r_5, r_6, r_7 win to move. Hence, as shown in Fig. 8.5(b), the system reaches c_2. At this configuration, r_3, r_5, r_6, and r_7 can move, but only r_3 moves one step forward because it wins the negotiation. In the following evolution, r_4 gets the priority to move first, and then r_5, r_6, and r_7. Thus, the system reaches c_3, as shown in Fig. 8.5(c). Currently, r_4 is blocked by r_1. When the movable robots r_3, r_5, r_6, and r_7 move one step forward again, the system is at c_4, as shown in Fig. 8.5(d). From this configuration, r_3 cannot

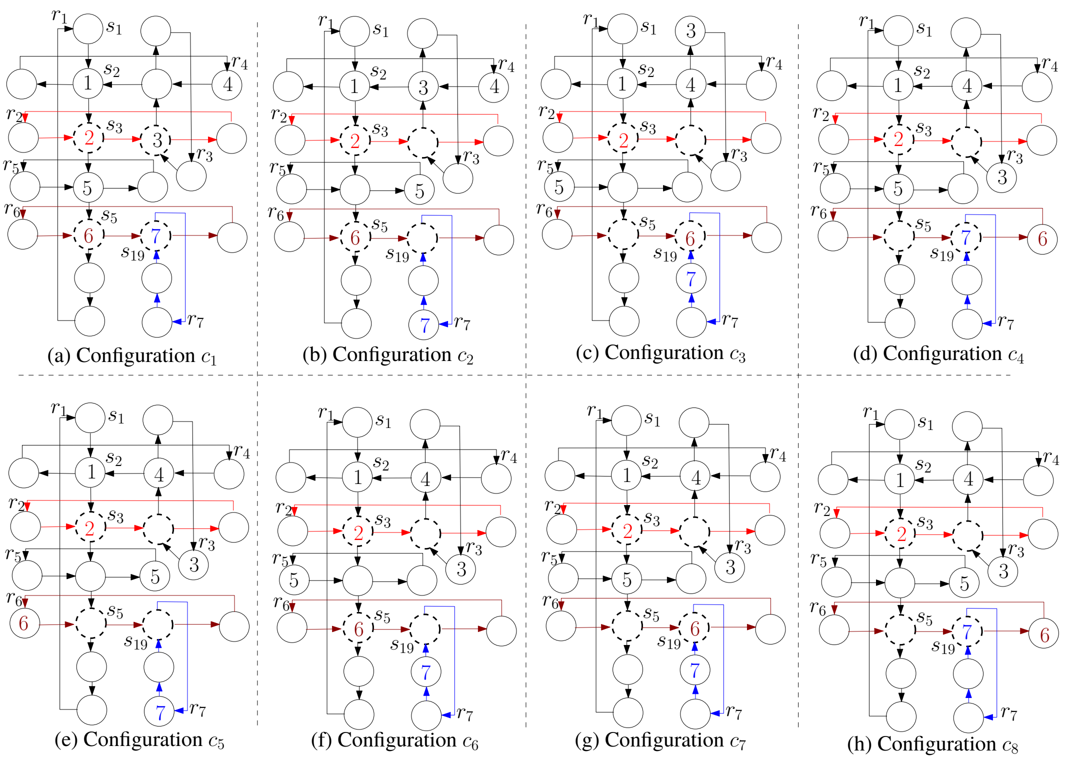


FIG. 8.5: System evolution without robust control algorithm. (a) r_2 is broken at s_3. (b) r_1 is blocked. (c) r_4 is blocked. (d) r_3 is blocked.

move forward anymore since its move can cause a deadlock with r_1, r_2, and r_4. Thus, after the system reaches c_4, r_1, r_2, r_3, and r_4 cannot move forward anymore. This fact can also be observed from the following evolution of the system shown in Figs. 8.5(e), 8.5(f), 8.5(g), and 8.5(h). In conclusion, when r_2 fails at s_3, we have $S^1_{2,s_3} = \{r_1\}$ and $S^\Delta_{2,s_3} = \{r_3, r_4\}$. Hence, the system is non-robust. The video of the simulation can be found at `https://youtu.be/xk1kAU-pQM0`.

8.4.2 Robot Motion with Robustness Algorithms

Next, we simulate the system with the governance of our robustness algorithms. From the simulation, all robots can move persistently except the robots that are directly dependent on the failed one.

Robot r_2 fails at s_3. First, all robots can move. After the negotiation, suppose r_1 gets the right to move forward and then moves one step forward. Based on Algorithm 8, $Flag(s_3, 1) = 1$ and $Flag(s_5, 1) = 1$ when r_1 reaches s_2. Based on Lines 12 and 13 of Algorithm 9, r_2 and r_6 cannot move to s_3 and s_5. Thus, the set of movable robots

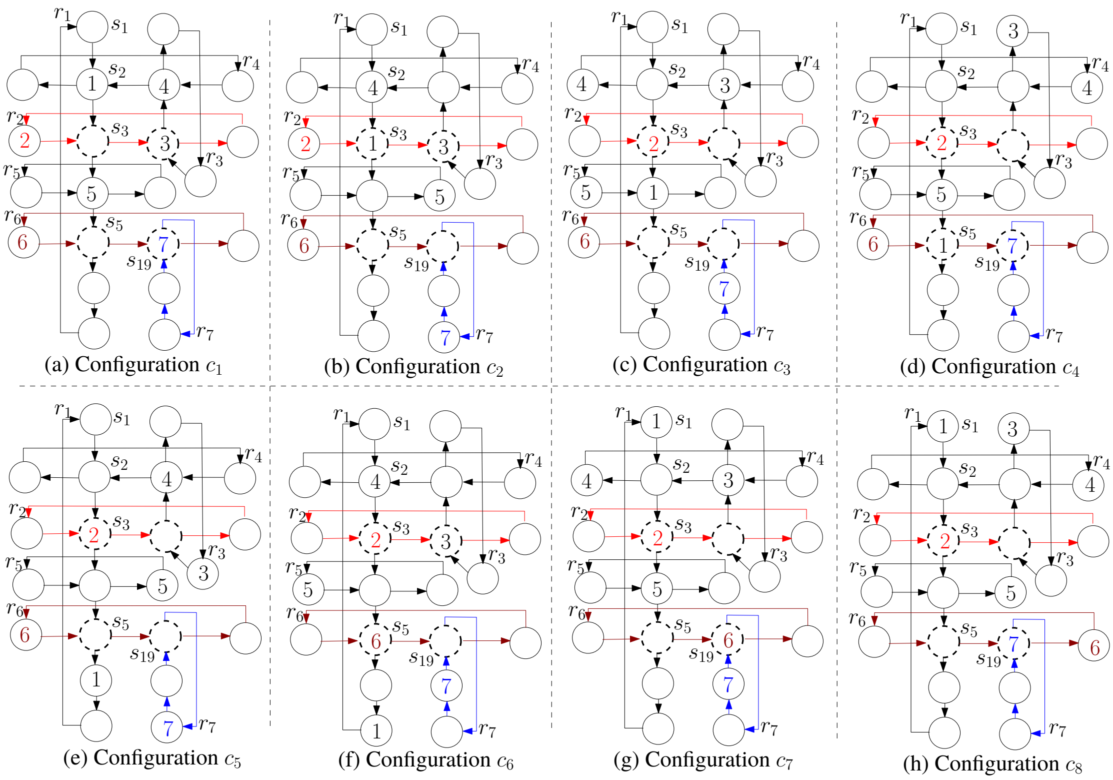


FIG. 8.6: System evolution under robust control algorithm. (c) r_2 fails at s_3. (g) r_1 cannot move anymore.

is $enable = \{r_1, r_3, r_4, r_5, r_7\}$. Suppose in the following negotiations, r_3, r_4, r_5, and r_7 get the right to move forward, respectively. Hence, the system reaches configuration c_1, as shown in Fig. 8.6(a). At c_1, $enable = \{r_1, r_5, r_7\}$. If $NEG(enable) = r_1$, r_1 moves to s_3, updating $enable$ to $\{r_1, r_4, r_5, r_7\}$. Suppose r_4, r_5, and r_7 get the right to move in the following negotiations. Thus, the system reaches c_2, as shown in Fig. 8.6(b). At c_2, r_1, r_3, r_4, r_5, and r_7 can move forward. Suppose r_1 is selected to move forward. When r_1 moves forward, $Flag(s_3, 1) = 0$. Hence, the current set of movable robots is $\{r_1, r_2, \ldots, r_5, r_7\}$. With the negotiation, r_2 is selected to move forward. When it reaches s_3, r_2 fails and cannot move anymore. When r_3, r_4, r_5, and r_7 are selected to move one step forward in the following negotiations, the system reaches c_3, as shown in Fig. 8.6(c). At c_3, r_1, r_3, r_4, and r_7 are able to move. After the negotiation, r_1 moves one step forward. As a result, r_1, r_3, r_4, r_5, and r_7 are able to move. Suppose r_3, r_4, r_5, and r_7 are selected to move one step forward. Then, the system reaches c_4, as shown in Fig. 8.6(d). When the system is at c_4, r_1, r_3, r_4, r_5, and r_7 are able to move forward. Suppose r_1 is selected to move after the negotiation. When r_1 moves one step forward, $Flag(s_5, 1) = 0$. However, r_6 still cannot move since the unreliable robot r_7 is on its path. Thus, the set of movable robots is $\{r_1, r_3, r_4, r_5, r_7\}$. Suppose r_3, r_4, r_5, and r_7 are selected to move one step forward, and thus, the system reaches c_5, as shown in Fig. 8.6(e). As shown in Figs. 8.6(f) and 8.6(g), the system next traverses c_6 and c_7 based on a sequence of negotiations. When the system reaches configuration c_7, r_1 cannot move forward anymore based on Lines 11 and 12 in Algorithm 8. With the movement of $r_3, r_4, \ldots, r_7$, the system reaches c_8, which is shown in Fig. 8.6(h). Clearly, r_1 does not block other robots' motion. Hence, the system is robust. The video of the simulation can be found at `https://youtu.be/xk1kAU-pQM0`.

8.4.3 Simulation Results on a Real Scenario

In this section, let us still consider the simulation abstracted from the real scenario shown in Fig. 6.14. We further suppose vehicle 1 is an unreliable robot. When it moves into the intersection, vehicle 1 fails, as shown in Fig. 8.7(a). At this time, vehicle 2 arrives at the intersection, but it cannot move into the intersection based on our proposed algorithm and then stops, as shown in Fig. 8.7(b). Next, vehicles 3 and 4 can move into the intersection since there are no collisions or deadlocks, as shown in Figs. 8.7(c) and 8.7(d). Continue their motion, vehicles 3 and 4 finally move away from the intersection successfully, as shown in Figs. 8.7(e) and 8.7(f), respectively.

8.5 Conclusion and Discussion

Herein, we study robust control of systems with unreliable robots, where robustness means that a failed robot can block the minimum number of robots in the system. Two distributed robustness algorithms are proposed for various robots to guarantee the robustness of the system. In addition to the theoretical analysis of the proposed robustness algorithms, experimental simulations are demonstrated. The results also validate the correctness of our approach.

The proposed concept of robustness is universal. Though the robustness we discussed in this work is for a system in which each robot has a predetermined path, it is

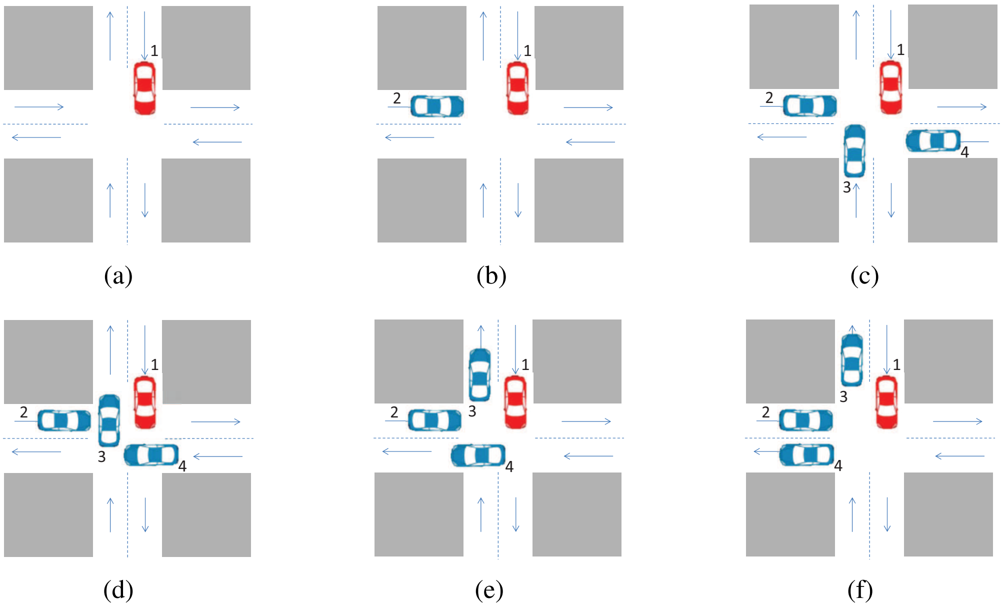


FIG. 8.7: Simulation results of the real scenario. (a) Vehicle 1 failed at the intersection; (b) Vehicle 2 cannot move into the intersection; (c) Vehicle 3 moves into the intersection; (d) Vehicle 4 can also move into the intersection; (e) Vehicle 3 moves away from the intersection; (f) Vehicle 4 pass through the intersection.

also adaptable in systems where a robot has multiple paths and can reroute its motion among these paths. In practice, even in such systems, a robot usually has only one route to move along in some areas. Besides, there may exist the following scenario: A robot at a state has two paths to select; the selection of the first one will cause a deadlock, while there is an unreliable robot on the second path; with the proposed robust control, the robot may not move forward. Thus, the proposed robustness can be widely used, especially in the systems where robots have finite paths to move along. In the future, we will conduct a detailed investigation of such scenarios.

Chapter 9

Hybrid Approach to Distributed Motion Control for Multi-Robot Systems

In Chapter 4, we study motion control from the low-level continuous control, which can generate continuous inputs to robots directly. However, it is hard to deal with deadlocks in some systems such as those studied in Chapters 6 and 7. While as shown in Chapters 6 and 7, high-level discrete models can avoid deadlocks efficiently, but cannot generate continuous inputs acting on robots' actuators. Hence, in this chapter, combining the discrete and continuous technologies studied in the previous chapters, we focus on distributed hybrid motion control for the system with fixed paths.

9.1 Introduction

Most of the current approaches are either discrete or continuous. On one hand, discrete methods usually abstract either the environment to a set of discrete states or the motion of a robot to a set of discrete actions; based on the abstraction, a robot can determine a sequence of discrete states or actions to execute. This kind of methods can simplify the motion control problem, but the main drawback is that no robot kinematics or dynamics are considered, and thus cannot provide direct inputs, e.g., accelerator or torque, to robot actuators. On the other hand, continuous methods usually depend on a robot's kinematic and/or dynamic equations and constraints. This kind of methods considers the environment as a continuous Euclidean space, and generates continuous paths or trajectories as well as the control inputs of actuators. However, for some complex environment or large number of robots, the computation cost will be high; moreover, few of these methods can deal with deadlocks efficiently.

To leverage the advantages of both discrete and continuous methods so as to not only deal with deadlocks efficiently but also obtain control inputs to actuators of robots, we focus on a hybrid approach to motion control of multi-robot systems where each robot has a predefined closed path to make persistent motion. It combines discrete supervisory control with continuous optimal control. For each local controller, based on its transition system and the equal-length partition, on each receding horizon, an online supervisory control policy first predicts whether the firing of its current transition would cause collisions or deadlocks. In case that the current transition cannot fire because of collisions or deadlocks, the robot will retrieve the robots it needs to wait for, as well as their current states. Second, the continuous control component is designed to compute optimal speed. It first predicts the motion time spent by other robots to resolve collisions or deadlocks; then considering this time constraint, the robot builds a local optimization problem and computes an optimal speed such that the robot can move to the next state as smoothly as possible. In the proposed approach, each robot only needs to communicate with its neighbors to retrieve immediately obtained information and hence can move in a fully distributed way. The communication protocols are described in Petri nets, and communication network can be reconfigured in real time based on the connectivity among robots. The simulation results show the effectiveness of our approach.

The contributions of this work are:

- We propose a fully distributed method for motion control of multi-robot systems where each robot has a predefined path. Each robot controls its motion only via communicating with its neighbors to retrieve some information that can be obtained immediately. Hence, robots can move in a fully distributed way.

- A local hybrid controller combining discrete and continuous control is designed for each robot. By discretizing a path into discrete states, the controller can deal with deadlocks as well as reduce the scale of the built local optimization problem; via continuous control based on mathematical programming and SCP, the controller can compute optimal speed to move.

- Communication protocols among robots are modeled by Petri nets. With the proposed communication protocols, a robot can adapt its communication to different neighbors during its motion. This guarantees the flexibility of the communication network and thus the scalability of the system.

The chapter is organized as follows. Section 9.2 gives the problem statement; Section 9.3 investigates the design and implementation details of the distributed hybrid approach; Section 9.4 models communication protocols using Petri nets for the proposed approach; Section 9.5 describes the experiment simulations, and Section 9.6 concludes the chapter.

9.2 Problem Statement

In this section, we first give more descriptions on the system we study and then state the problem we focus on.

We assume that each path $p = p(\theta)$ is continuously differentiable. Indeed, the path of a robot is the geometric curve of its trajectory. Based on physical laws, at any time instant, the gradient of a trajectory at a position equals to the instant velocity at this location. Since velocity function is continuous, the gradient of a path should be continuously differentiable. Hence, the assumption is reasonable in real world. Note that even though sometimes the given reference path is not continuously differentiable, the real generated path, which is around the reference one, should be continuously differentiable. If a path is not continuously differentiable, we can first use a continuously differentiable path to approximate it using some methods, such as pure pursuit algorithms.

Definition 33. The speed of a robot is a scalar function with respect to time, mapping from $\mathbb{R}_0^+$ to $\mathbb{R}_0^+$, where $\mathbb{R}_0^+$ is the set of nonnegative real numbers.

Note that if the speed of a robot at time instant t is denoted as $v(t)$, which is a scalar variable, and the velocity is denoted as $\mathbf{v}(t)$, which is a vector, then we have $v(t) = \|\mathbf{v}(t)\|_2$.

Each robot is required to move along its path persistently without causing any collision and deadlock. Since the path is determined, the motion direction at each point for a robot is fixed and we can guarantee motion safety only by controlling motion speed. Hence, the motion control problem in our case can be described as follows.

Problem 5. Given a set of closed paths for robots in a multi-robot system, determine proper speed for each robot such that robots can move along their own paths as smoothly and quickly as possible without causing any collision or deadlock.

As described in previous chapters, high-level discrete control is an efficient way to avoid collisions and deadlocks in such a system. However, they cannot deal with speed optimization of robots directly. This may cause robots perform sharp stops with very high deceleration, and a robot may stop and resume its motion frequently (will give details in our experiments). Clearly, it is not a desired motion and is energy-costly. Hence, in this chapter, we investigate a hybrid approach to robot motion. A discrete control policy based on state transition systems is applied to avoid collisions and deadlocks, and an optimal speed control strategy is used to deal with continuous motion at each state. In the following section, we give the details of our approach.

9.3 Hybrid Approach to Motion Control

In this section, we describe a hybrid approach to solving the problem. The control architecture of our approach is shown in Fig. 9.1. The motion controller obtains the inputs from its sensors and neighbors, then computes the control inputs and feeds back to the actuator. The control process contains two phases. The first one is discrete motion control. At this phase, the robot can build and refine its discrete model by detecting the environment, i.e., the path network of the system, and then determines transition firing to avoid collisions and deadlocks by communicating with its neighbors. The second one is continuous motion control at each discrete state. Based on the discrete

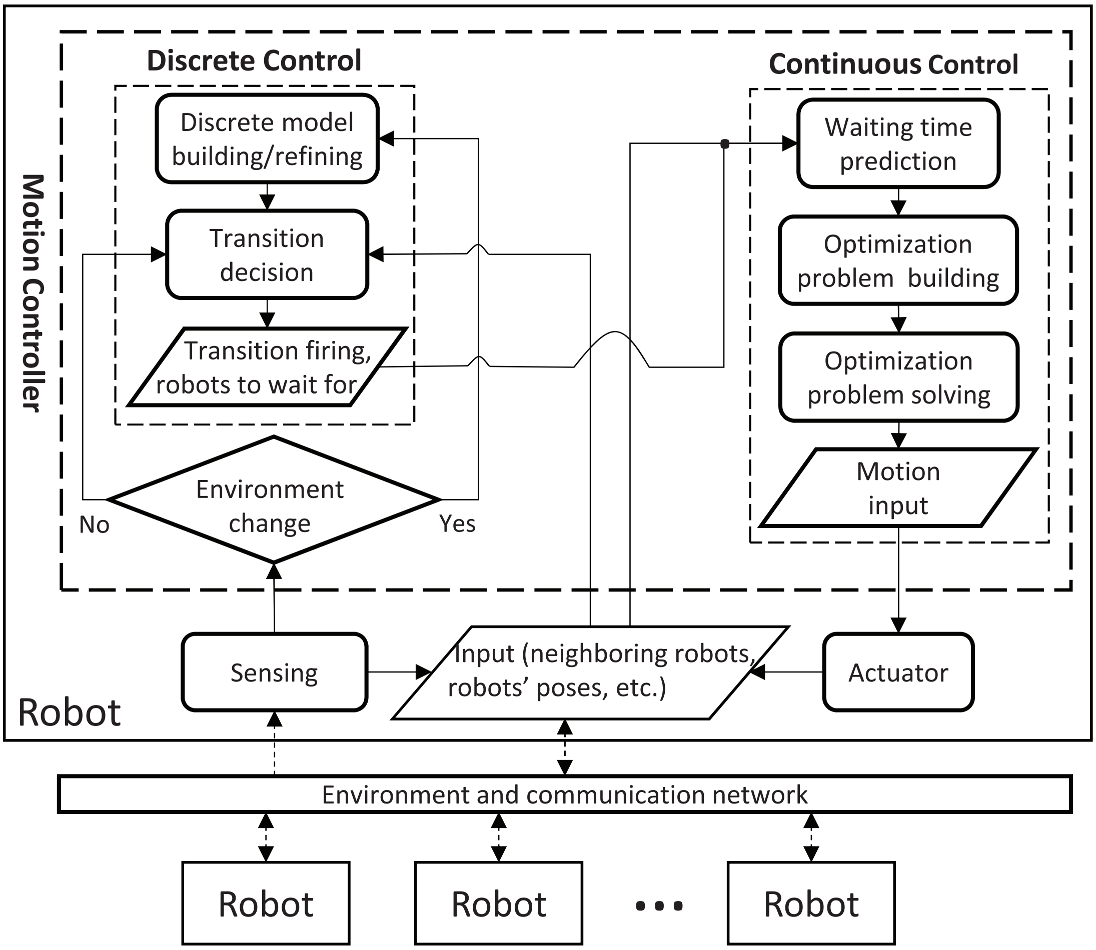


FIG. 9.1: Framework of the proposed hybrid motion control approach.

decision obtained at the first phase, the robot predicts the time it should wait at the current state, and then computes proper acceleration for the actuator by building and solving an optimization problem. In the sequel, Sections 9.3.1 and 9.3.2 describe the related discrete control via transition systems and speed optimization with mathematical programming, and Section 9.3.3 verifies the effectiveness of the proposed approach.

Before describing the details, we give some assumptions. As shown in Fig. 9.1, the accurate motion of a robot relies on many aspects, such as the controller, actuator, sensor, and communication network. Since we focus on the motion controller, we assume that other components can always perform well. For example, the actuator can respond correctly to its inputs, the sensors can always work well to monitor the environment correctly, and the communication network among robots can transmit messages without loss and delay.

9.3.1 Discrete Transition Control

To deal with deadlocks and obtain continuous control inputs, as well as guarantee motion flexibility, we focus on hybrid control. In this subsection, by discretizing a path and building a transition system, we describe discrete control of a robot. In the next subsection, we consider continuous control, where each time we only focus on the continuous motion at the current state, rather than plan the motion for the whole path at once.

Based on Chapter 5, the transition system of r_i is a tuple $\mathcal{T}^i = \langle S^i, T^i \rangle$, where $T^i = \rightarrow_{i,move}$. Recall that $S^i = S^i_\alpha \cup S^i_\beta$, where S^i_α and S^i_β are sets of collision and private states, respectively; $\forall s \in S^i$, $Pre_i(s)$ and $Pos_i(s)$ denote the preceding and succeeding states of s in S^i, respectively, i.e., $(Pre_i(s), s) \in T^i$ and $(s, Pos_i(s)) \in T^i$. Suppose $s_{cur,i}$ is the current state of r_i, then $(s_{cur,i}, Pos_i(s_{cur,i}))$ is the *current transition* of r_i. Hence, the task at the discrete control phase is to determine whether r_i's current transition can fire. In the sequel, we develop an algorithm to make a decision on whether the current transition of a robot can be fired based on its transition system.

First, with the definitions of collision and deadlocks in Definitions 11 and 12 in Chapater 6, we have:

Definition 34. Suppose r_i is at s. Its current transition $(s, Pos_i(s))$ is enabled if there are no collisions and deadlocks when r_i is at $Pos_i(s)$. A transition can fire if and only if it is enabled.

Similar with the analysis in Chapter 6, to avoid collisions, each robot stores a set of local signals which denote the status of its collision states. Let $Sign_i$ denote the set of signals identifying the status of collision states in S^i_α. $\forall s \in S^i_\alpha$, $Sign_i(s) = 1$ if s is occupied by other robots; otherwise, $Sign_i(s) = 0$. If the next state is a collision state and its signal is 1, then the transition cannot be enabled. To avoid deadlocks, a robot needs to communicate with its neighbors to check whether there exists any deadlock cycle. Recall the process as follows. Suppose r_i is at s and $Sign_i(Pos_i(s)) = 0$. To check whether $(s, Pos_i(s))$ can be enabled, r_i should further check whether there would be any deadlock if it was at $Pos_i(s)$. First, r_i checks the state $Pos_i(Pos_i(s)) \triangleq cs_i$. If cs_i is occupied by a robot r_{j_1}, then r_{j_1} checks the status of $Pos_{j_1}(cs_i) \triangleq cs_{j_1}$. Similarly, if cs_{j_1} is occupied by another robot r_{j_2}, then r_{j_2} checks the state $Pos_{j_2}(cs_{j_1})$.

Continue this procedure until there exists a robot such that its next state is $Pos_i(s)$ or is not $Pos_i(s)$ and not occupied. The former means there exists a deadlock, while the latter means no deadlocks can occur at $Pos_i(s)$ and the transition $(s, Pos_i(s))$ is enabled. The procedure of deadlock detection corresponding to $Pos_i(s)$ is denoted as $Dect(r_i, Pos_i(s))$. $Dect(r_i, Pos_i(s)) = 0$ means that there is no deadlock, while $Dect(r_i, Pos_i(s)) = k > 0$ means that a deadlock is detected and r_k is the last one in the circuit, i.e., $Pos_k(s_{cur,k}) = Pos_i(s)$, where $s_{cur,k}$ is the current state of r_k.

Since different robots may make decisions at the same time, simultaneous transition firing may cause conflicts. Hence, an enabled transition may not really fire. Indeed, the related robots need to negotiate with each other to determine whose transition can finally fire. Note that different with Chapter 6, where a robot only needs to determine whether it can move or not, in this chapter, if a robot cannot fire its current transition, it should further predict the motion time at this state. Hence, before giving the negotiation process, we need to introduce some definitions.

Definition 35 (Path Length). Suppose x_0 and x are two points on a path. The path length from x_0 to x, denoted as $l(x_0, x)$, is the length of the path segment from x_0 to x along with the motion direction.

Given a path $p(\theta)$, $l(x_0, x)$ can be computed as $l(x_0, x) = \int_{\theta_0}^{\theta_1} \| \frac{dp(\theta)}{d\theta} \|_2 d\theta$, where $x_0 = p(\theta_0)$ and $x = p(\theta_1)$. Given p^i and S^i of r_i, its path segment from x to y is denoted as $l_i(x, y)$; $L_i(s)$ and $x_{i,s}$ denote the length and the end point of path segment of p^i represented by s, respectively.

Definition 36 (Hybrid State). The hybrid state of a robot r_i is a quadruple (s_i, x_i, v_i, Lr_i), where $s_i \in S^i$, $x_i \in p^i$ is a position on the path segment of s_i, v_i is the speed at x_i, and Lr_i is the path length from x_i to the end of s_i.

Suppose X is the negotiation region that may cause conflicts due to simultaneous motion of multiple robots. At any time instant, the robots that are movable into/in X, denoted as E_X, should communicate to determine the robots that can finally fire their current transitions. The main idea for the negotiation is that the robot with shorter time to its next state can check and make a decision first, and others make their decisions based on the decisions made by the previous robots. The detailed algorithm is shown

Algorithm 10: Negotiation process to avoid conflicts.

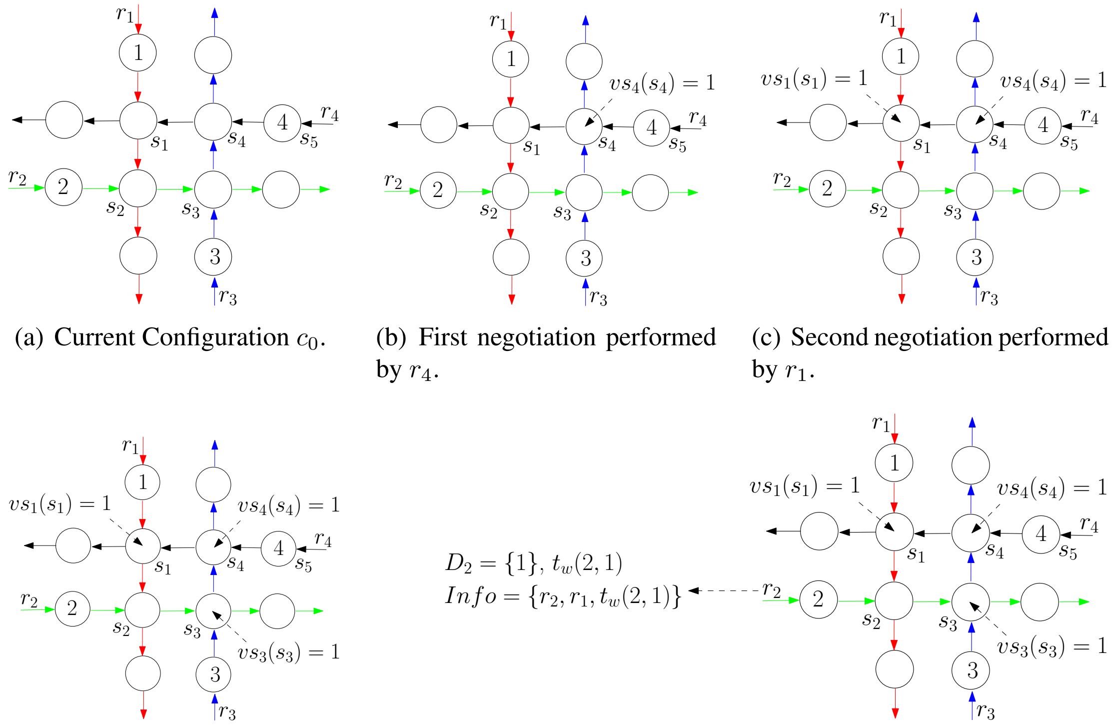
Input : Movable robots E_X, and their current hybrid states (s_i, x_i, v_i, Lr_i) and signals $Sign_i$.

Output: MV and UM: robots that can fire and cannot fire, respectively; $Info = \{(r_i, r_j, t_w(i,j)), i \in UM\}$, where r_i needs to wait for r_j and the predicted waiting time is $t_w(i,j)$.

1 Initialization: $vs_i = Sign_i, \forall i \in E_X$; $MV = \emptyset$; $UM = \emptyset$; $Info = \emptyset$;
2 Compute time to the next state: $t_i = Lr_i / v_i, \forall i \in E_X$;
3 **while** $E_X \neq \emptyset$ **do**
4     $k = \arg\min_{i \in E_X} t_i$; $s = Pos_k(s_k)$;
5   **if** $\exists j \in E_X$ such that $vs_j(s) = 1 \parallel Dect(r_k, s) = j$ based on vs **then**
        /* r_k will cause a collision or a deadlock if it is at s after the moves of robots in MV. */
6         $UM = UM \cup \{k\}$; $E_X = E_X \setminus \{k\}$;
7         $D_k = j$; $t_w(k,j) = l_j(x_j, x_{j,s}) / v_j$;
8         $Info = Info \cup \{(r_k, r_j, t_w(k,j))\}$;
9   **else**
10         $MV = MV \cup \{k\}$; $E_X = E_X \setminus \{k\}$;
11         $vs_k(s) = 1$;

in Algorithm 10. First, each robot predicts its time to arrive at its next state (Line 2). This information is broadcast to robots in E_X. Then, these robots check one by one to determine whether they can fire their current transitions (Lines $3 - 11$) based on the temporary signals $vs_i, i \in E_X$, whose initial values are equal to $Sign_i$. Suppose among the remaining robots in E_X, r_k is the robot with the shortest arriving time to its next s-tate. Based on the temporary signal sv_i, r_i checks whether its motion to $s = Pos_i(s_{cur,i})$ causes any collision or deadlock after the firings of the former robots' current transition-s. If "yes", r_k is not allowed to fire its current transition when it reaches the end of the state, so it computes the robots to be waited for and the corresponding waiting time (Lines 5−8), which will be used in the continuous speed control. Otherwise, r_k is al-lowed to fire its current transition and change its temporary signal, i.e., $sv_k(s) = 1$ (Line 11). Once a robot has checked its motion, the robot is removed from E_X.

Fig. 9.2 shows an example to illustrate the negotiation process. At the current configuration shown in Fig. 9.2(a), all local temporary signals in $vs_i, i = 1, 2, 3, 4$, are 0, and the predicted motion time to the end of these robots' current states are $t_1 = 1, t_2 = 1.2, t_3 = 1.1$, and $t_4 = 0.9$ (Line 2). Hence, r_4 performs the first iteration of the



(a) Current Configuration c_0.

(b) First negotiation performed by r_4.

(c) Second negotiation performed by r_1.

(d) Third negotiation performed by r_3.

(e) Fourth negotiation performed by r_2.

FIG. 9.2: An example to illustrate the negotiation process. (a) The current configuration for negotiation, and $t_1 = 1, t_2 = 1.2, t_3 = 1.1, t_4 = 0.9$; (b) r_4 starts to perform the first iteration of negotiation and it can move forward, causing $vs_4(s_4) = 1$; (c) r_1 performs the second negotiation and determines that it is movable, and changes $vs_1(s_1) = 1$; (d) the third iteration is done by r_3 and r_3 determines its move and sets $vs_3(s_3) = 1$; (e) at the fourth iteration of the negotiation, r_2 cannot move forward based on $vs_4(s_4)$, $vs_3(s_3)$, and $vs_3(s_3)$, and needs to wait for the move of r_1.

negotiation, i.e., Lines 5 − 11, based on Line 4 in the algorithm. Since $vs_3(s_4) = 0$ and $vs_1(s_1) = 0$, r_4 executes Lines 9 − 11, causing $vs_4(s_4) = 1$, as shown in Fig. 9.2(b). At this moment, $MV = \{4\}$. Second, r_1 executes the second iteration. Similarly, r_1 finds that it can move one step forward, resulting in $vs_1(s_1) = 1$ and $MV = \{1, 4\}$, as shown in Fig. 9.2(c). Third, as shown in Fig. 9.2(d), r_3 begins the third iteration. Currently, $sv_2(s_3) = 0$ and $Dect(r_3, s_3) = 0$. Hence, r_3 is allowed to move and then $vs_3(s_3) = 1$ and $MV = \{1, 3, 4\}$. At last, r_4 needs to check whether it can move one step forward based on the former negotiations. Since $sv_3(s_3) = sv_4(s_4) = sv_1(s_1) = 1$, $Dect(r_2, s_2) = 1$, meaning that a deadlock will occur if r_2 also moves to s_2 and r_2 needs to wait for r_1 moving away from s_2. Hence, as shown in Fig. 9.2(e), the fourth iteration results in $D_2 = 1$, $UM = \{2\}$, and the waiting time $t_w(2, 1) = l_1(x_1, x_{1,s_2})/v_1 = (Lr_1 + L_1(s_1) + L_1(s_2))/v_1$ based on Lines 6 − 8 in Algorithm 10.

Algorithm 11: Decision for transition firing of r_i.

Input : Discrete model $\mathcal{T}^i$, current state s_i, and negotiation region X.

Output: $decision = 0$: r_i cannot fire its current transition due the occurrence of collisions or deadlocks; $decision = 1$: r_i cannot fire its current transition due to the negotiation process; and $decision = 2$: r_i can fire its current transition.

1 $MV = \emptyset, UM = \emptyset, Info = \emptyset$;
2 **if** $Pos_i(s_i) \in S^i_\beta$ **then**
3     $decision = 2$;
4 **else if** $Sign_i(Pos_i(s_i)) = 1$ **then**
    `/*` r_i`'s motion will cause a collision. */`
5     $decision = 0$;
6 **else**
7   **if** $Pos_i(Pos_i(s_i)) \in S^i_\alpha$ & $Dect(r_i, Pos_i(s_i)) > 0$ **then**
        `/*` r_i`'s motion will cause a deadlock. */`
8         $decision = 0$;
9   **else**
10         $(s_i, Pos_i(s_i))$ is enabled and add i to E_X;
11         $\{MV, UM, Info\}$ = Algorithm 10;
12   **if** $r_i \in MV$ **then**
            `/*` r_i `can fire its current transition. */`
13             $decision = 2$;
14   **else**
            `/*` r_i `cannot fire its current transition. */`
15             $decision = 1$
16 **return** $\{MV, UM, Info, decision\}$;

Based on the above analysis, Algorithm 11 shows the procedure of robot r_i to determine whether its current transition can actually fire. If the next state is a private state, the current transition can always be enabled and fired (Lines 2 and 3). However, if the next state is occupied by another robot, the current transition cannot be enabled (Lines 4 and 5). In other cases, if a deadlock is detected, the current transition cannot be enabled (Lines 7 and 8); otherwise, r_i needs some negotiation to determine whether its transition can fire (Lines $10 - 15$).

Algorithm 11 focuses on the discrete decision in order to avoid collisions, deadlocks, and conflicts. Next, we describe the procedure for continuous speed control.

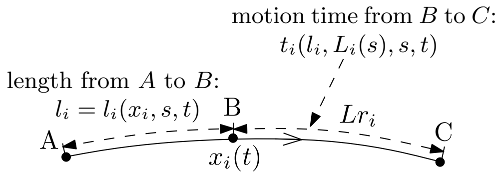


FIG. 9.3: An illustration of notations related to discrete state and continuous path. Arc $\widehat{ABC}$ is the path segment abstracted to s, where A is the start of s and C is the end of s. $B(x_i(t))$ is the position of r_i at t; $l_i(x_i, s, t)$ is the arc length of $\widehat{AB}$; Lr_i is the arc length of $\widehat{BC}$; $L_i(s) = l_i(p_i, s, t) + Lr_i$ is the length of $\widehat{ABC}$; and $t_i(l_i, L_i(s), s, t)$ is the motion time on the arc $\widehat{BC}$.

9.3.2 Continuous Speed Adjustment

In this subsection, we describe the algorithm for speed adjustment of a robot when its current transition cannot fire. Some notations are used during our descriptions. Given a discrete state $s \in S^i$ where r_i is at time instant t, suppose $x_i(t)$ is the position on the path segment of s at t, the path length from the start of s to $x_i(t)$ is $l_i(x, s, t)$. The speed and acceleration at $x_i(t)$ are $v_i(l_i, s, t)$ and $a_i(l_i, s, t)$, respectively. Motion time from $x_i(t)$ to the end of s is $t_i(l_i, L_i(s), s, t)$. Recall that $L_i(s)$ is the length of r_i's path segment of s. Fig. 9.3 shows an illustration of these notations. For simplicity and without ambiguity, we omit s and t in the notations during our discussion. Hence, the path length of s is L_i; the speed and acceleration at $x_i(t)$ are $v_i(l_i)$ and $a_i(l_i)$, respectively; motion time from $x_i(t)$ to $x_{i,s}$ (the end of s) is $t_i(l_i, L_i)$.

In this chapter, real-time and distributed speed adjustment is performed based on the MPC strategy, where at each time instant, the speed of robot is computed from a local optimization problem. In the sequel, we introduce the construction of the local optimization problem of r_i at the current time instant t_0.

To build the distributed optimization problem, we first describe the kinematic equations of a robot. Suppose the current hybrid state of r_i is $(s, x_i(t_0), v_i(l_0), Lr_i)$, where $l_0 = l_i(x_i, s, t_0)$ is the path length from the start of s to $x_i(t_0)$, then its kinematic equations can be described as follows.

$$t_i(l_0, L_i) = \int_{l=l_0}^{l=L_i} \frac{1}{v_i(l)} dl, \tag{9.1}$$

$$\frac{1}{2}v_i^2(L_i) - \frac{1}{2}v_i^2(l_0) = \int_{l=l_0}^{l=L_i} a_i(l) dl, \tag{9.2}$$

where $t_i(l_0, L_i)$ is the motion time from the current position $x(t_0)$ to the end of s.

Indeed, we have

$$v_i(l) = \frac{dl}{dt_i(l)}.$$

This means

$$dt_i(l) = \frac{1}{v_i(l)}dl.$$

Hence, according to the theory of integral, we have

$$\int_0^{t_i(l_0,L_i)} dt_i = \int_{l_0}^{L_i} \frac{1}{v_i(l)}dl,$$

which generates (9.1).

Since

$$a_i(l) = \frac{dv_i(l)}{dt_i(l)} \text{ and } dt_i(l) = \frac{dl}{v_i(l)},$$

we have

$$a_i(l) = \frac{v_i(l)dv_i(l)}{dl} = \frac{1}{2}\frac{d[v_i^2(l)]}{dl}.$$

This implies (9.2).

In the sequel, we give the procedure to build the local optimization of a robot.

First of all, in case that its current transition cannot fire, a robot needs to predict the time that other robots spend in passing through their required states. There are two situations that a robot cannot fire its current transition. The first one is that the current transition is enabled but cannot fire since simultaneous firing causes a conflict, and the second one is that the current transition cannot be enabled.

We first propose an algorithm to compute waiting time for the former situation. The main process is that a robot r_i may need to wait for the move of a robot r_j in UM, then r_j also needs to wait for the move of another robot in UM. Continue the process until the waited robot is in MV. Algorithm 12 shows the procedure for r_i to compute its waiting time based on the negotiation process. At first, r_i checks the robot, say r_j, that it needs to wait for (Line 1). Then, r_j further checks the robot it needs to wait

Algorithm 12: Computation of r_i's waiting time during its negotiation process.

Input : MV, UM, and $Info$ based on Algorithm 10.

Output: Waiting time $tw(i)$.

1 $Info_i = (r_i, r_j, t_w(i,j))$;

2 $tw(i) = t_w(i,j)$; /* r_i needs to wait for the former robot r_j in the negotiation process. */

3 $i_1 = j$;

4 **while** $i_1 \in UM$ **do**

5     $Info' = (r_{i_1}, r_{i_2}, t_w(i_1, i_2))$; /* r_{i_1} needs to wait for r_{i_2} based on the negotiation process */

6     $tw(i) = \max\{tw(i), t_w(i_1, i_2)\}$;

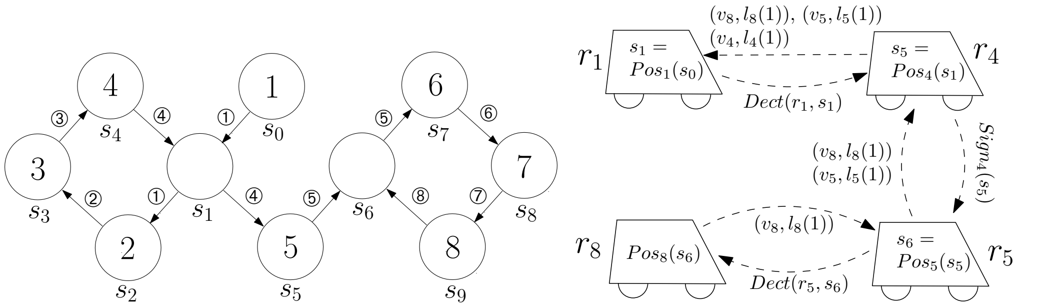
7     $i_1 = i_2$;

for. Iteratively, the procedure stops when a robot can fire its current transition, i.e., the while-loop in Lines 4–7 .

Next, we describe the computation of waiting time in the second situation. To enable its current transition, a robot depends on the move of some other robots, called enable-dependent robots.

Definition 37. The set of enable-dependent robots of r_i at state s, denoted as $D_i(s)$, is a set of robots that r_i needs to wait for in order to enable its current transition $(s, Pos_i(s))$.

A robot r_i can determine its dependent robots $D_i(s)$ via a sequence of communication. First, in order to check whether it can move to $Pos_i(s) \triangleq cs_i$, r_i determines the robot it needs to directly wait for based on $Sign_i(cs_i)$ and $Dect(r_i, cs_i)$. There are two cases: (1) If it finds cs_i is occupied by a robot r_{j_1}, r_i sends a message, including the information of cs_i, to r_{j_1} to inform that r_{j_1} needs to move to $Pos_{j_1}(cs_i) \triangleq cs_{j_1}$. Thus, when r_{j_1} receives this message, it checks whether it can move to cs_{j_1}. (2) If r_i receives a message from robot r_{j_1} during $Dect(r_i, cs_i)$ and identifies a deadlock, then r_i sends the checked result and the information of cs_i to r_{j_1}, and r_{j_1} begins to check whether it can move to cs_{j_1}. This means the deadlock can only be resolved when r_{j_1} moves to cs_{j_1}. Hence, in both cases, r_{j_1} receives a message from r_i and needs to check, if needed, whether it can arrive at cs_{j_1}. This is done via checking $Sign_{j_1}(cs_{j_1})$ or executing $Dect(r_{j_1}, cs_{j_1})$. Like r_i, r_{j_1} can retrieve the robot it needs to wait for, say r_{j_2}, and sends a notification message to r_{j_2}. Note that r_{j_2} is also an enable-dependent robot of r_i. Similarly, after receiving the message, r_{j_2} needs to check whether it can



(a) Current configuration. (b) Retrieval of enable-dependent robots.

FIG. 9.4: An illustration of enable-dependent robots and their retrieval. (a) The current configuration, where the arrows with ⓝ denote the transition of r_n; (b) The retrieval process, where dashes arrows denote the communication between robots.

move to $Pos_{j_2}(cs_{j_1}) \triangleq cs_{j_2}$ and retrieves the robot to wait for based on $Sign_{j_2}(cs_{j_2})$ or $Dect(r_{j_2}, cs_{j_2})$. One by one until there exists a robot that can move to the required state. In this way, r_i can retrieve the robots to wait for at s. During the sequence of communication, a robot also sends back its current speed and the path length required to move, which will be used by r_i to compute its waiting time.

For example, consider the configuration shown in Fig. 9.4(a). At the current time, r_1 is at s_0. By performing $Dect(r_1, s_1)$, r_1 receives a message from r_4 and identifies a deadlock. Hence, r_1 sends a notification message to r_4 and r_4 begins to check whether it can move to s_5 (i.e., $Pos_4(Pos_1(s_0))$). After checking $Sign_4(s_5)$, r_4 detects r_5 at s_5 (i.e., $Sign_4(s_5) = 1$) and then sends a message to r_5. r_5 needs to determine whether it can move to s_6 when it receives this message. Since s_6 is not occupied by any robots, r_5 performs $Dect(r_5, s_6)$. At the end of $Dect(r_5, s_6)$, r_5 receives a message from r_8 and identifies a deadlock. So r_5 notifies r_8. Assume that $Pos_8(s_6)$ is a private state. Then r_8 will send a message, including the path length needed to move, to notify r_5 that it can pass through s_6. Consequently, r_5 will send back to r_4 this information plus its required moving path length, and r_4 also sends the related information to r_1. Hence, r_1 retrieves its dependent robots $D_1(s_0) = \{r_4, r_5, r_8\}$, as well as their path lengths needed to move. Fig. 9.4(b) shows the retrieval process.

Based on the definition of enable-dependent robots, r_i cannot leave s until all robots in $D_i(s)$ arrives at the required states. Hence, r_i needs to predict its least motion time at s, which is computed as follows: $\forall r_j \in D_i(s)$, suppose its speed and path length needed to move are v_j and $l_j(i)$, then its predicted motion time is $pt_j = l_j(i)/v_j$. In the sequence of enable-dependent robots, the last robot robot, say r_k, is a movable robot. So its waiting time can be obtained via its negotiation process based on Algorithm 12. Thus, r_i's waiting time at s is $t_w(i) = \max\{pt_j \text{ for } r_j \in D_i(s), tw(k)\}$.

Based on the waiting time, we can now give the local optimization problem of r_i at the current time t_0, which is shown in (9.3). Recall that $l_0 = l_i(x_i, s, t_0)$ is r_i's path length from the start of s to its current position $x_i(t_0)$ and $v_i(l_0)$ is the current speed. (9.3a) is the objective function. In this work, we consider two requirements: move as smoothly as possible, which can guarantee stability and smoothness; and pass through the state as soon as possible, which can give way to others. Hence, the objective function contains two parts: the former is for motion performance and the latter for motion time, where w_1 and w_2 are weights. Constraint (9.3b) describes the kinematics of the robot. Constraints (9.3c) and (9.3d) describe the physical constraints of the robot. It is the inherent property of a robot. At last, constraint (9.3e) describes the constraint to avoid collisions and deadlocks, meaning that r_i cannot move into the next state before the waiting time $t_w(i)$.

$$\min_{a_i} w_1 \sqrt{\int_{l=l_0}^{l=L_i(s)} a_i(l)^2 dl} + w_2 \int_{l=l_0}^{l=L_i(s)} \frac{1}{v_i(l)} dl \tag{9.3a}$$

subject to: $\forall L \in [L_0, L_i(s)]$,

$$\frac{v_i(L)^2}{2} = \frac{v_i(l_0)^2}{2} + \int_{l=l_0}^{l=L} a_i(l) dl, \tag{9.3b}$$

$$0 \le v_i(L) \le v_{\max}, \tag{9.3c}$$

$$a_{\min} \le a_i(L) \le a_{\max}, \tag{9.3d}$$

$$t_w(i) \le \int_{l=l_0}^{l=L_i(s)} \frac{1}{v_i(l)} dl. \tag{9.3e}$$

To deal with the integral equations in the above problem, we would like to find a numerical solution. Hence, we first discretize the path segment of s: $0 = L_0, L_1, \ldots, L_K = L_i(s)$, where $h = L_i(s)/K$ and $\forall k \in \mathbb{I}_K = \{0, 1, \ldots, K\}$, $L_k = kh$. After discretization, computation only happens at each discrete point, so $\exists k_0 \in \mathbb{I}_K$ such that $l_0 = k_0 h$. Then, the control variable $a_i(L)$ is discretized via piecewise constant: $\forall L \in [L_k, L_{k+1})$, $a_i(L) = a_i(L_k)$. To guarantee safety always, the discrete step length should satisfy that it is possible to stop a robot between two successive steps in the worst case. This means that h should satisfy: $2|a_{\min}|h \geq v_{\max}^2$.

Let $b_i(L) = v_i(L)^2$, $b_i^k = b_i(L_k)$, and $a_i^k = a_i(L_k)$, $\forall k \in \mathbb{I}_K$. Substituting a_i^k into (9.3b), we have

$$b_i(L) = b_i^k + 2a_i^k(L - L_k), \forall L \in (L_k, L_{k+1}]. \tag{9.4}$$

This implies

$$\int_{l=L_k}^{l=L_{k+1}} \frac{1}{v_i(l)} dl = \int_{l=L_k}^{l=L_{k+1}} \frac{1}{\sqrt{b_i^k + 2a_i^k(l - L_k)}} dl = 2h/(\sqrt{b_i^{k+1}} + \sqrt{b_i^k}), \tag{9.5}$$

and

$$\int_{l=l_0}^{l=L_i(s)} \frac{1}{v_i(l)} dl = \sum_{k=k_0}^{K-1} \int_{l=L_k}^{l=L_{k+1}} \frac{1}{v_i(l)} dl = 2h \sum_{k=k_0}^{K-1} \frac{1}{\sqrt{b_i^{k+1}} + \sqrt{b_i^k}} \tag{9.6}$$

Hence, the optimization problem in (9.3) is reformulated as the following discretized form:

$$\min_{\mathbf{a}_i, \mathbf{b}_i} w_1 \sqrt{h} \|\mathbf{a}_i\|_2 + w_2 \sum_{k=k_0}^{K-1} \frac{2h}{\sqrt{b_i^{k+1}} + \sqrt{b_i^k}} \tag{9.7a}$$

subject to:

$$\mathbf{A}\mathbf{b}_i - 2h\, \mathbf{a}_i = 0 \tag{9.7b}$$

$$\mathbf{b}_i(1) = v_i^2(k_0) \tag{9.7c}$$

$$\mathbf{0} \leq \mathbf{b}_i \leq v_{\max}^2 \mathbf{1}, \tag{9.7d}$$

$$a_{\min}\mathbf{1} \leq \mathbf{a}_i \leq a_{\max}\mathbf{1}, \tag{9.7e}$$

$$t_w(i) - \sum_{k=k_0}^{K-1} \frac{2h}{\sqrt{b_i^{k+1}} + \sqrt{b_i^k}} \leq 0, \tag{9.7f}$$

where $\mathbf{b}_i = (b_i^{k_0}, b_i^{k_0+1}, \ldots, b_i^K)^T$ and $\mathbf{a}_i = (a_i^{k_0}, a_i^{k_0+1}, \ldots, a_i^{K-1})^T$ are variables; $\mathbf{A} = (A_{kj})_{(K-k_0)\times(K-k_0+1)}$ satisfies $\forall k = 1, 2, \ldots, K - k_0$, $A_{kj} = -1$ for $j = k$, $A_{kj} = 1$ for

$j = k+1$, and $A_{kj} = 0$ for others; $v_i(k_0)$ is the current speed; and **0** and **1** are vectors of zeros and ones with proper dimensions, respectively.

Remark 10. Note that in practice, the speed can be zero, meaning that the robot has to stop its motion to wait for others. To deal with this case, we assign a sufficiently large number, e.g., 10^6, to $1/v$ if $v = 0$.

Clearly the local optimization problem (9.7) is non-convex due to constraint (9.7f), which can be regarded as a difference between two convex functions. Next, we apply SCP to solve it approximately. In the sequel, we describe the detailed procedure (9.7).

Let $g(\mathbf{b}_i) = \sum_{k=k_0}^{K-1} \frac{2h}{\sqrt{b_i^{k+1}}+\sqrt{b_i^k}}$, and $\nabla g(\mathbf{b}_i)$ is the gradient of $g(\mathbf{b}_i)$. Its first-order Taylor approximation at a given point ${}^m\mathbf{b}_i$ can be described as $g({}^m\mathbf{b}_i) + \nabla g({}^m\mathbf{b}_i)^T(\mathbf{b}_i - {}^m\mathbf{b}_i)$. Hence, the convex approximation of (9.7) at ${}^m\mathbf{b}_i$, denoted as $P({}^m\mathbf{b}_i)$, can be described as follows.

$$\begin{aligned} \min_{\mathbf{a}_i,\mathbf{b}_i} \quad & w_1\sqrt{h}\|\mathbf{a}_i\|_2 + w_2 \sum_{k=k_0}^{K-1} \frac{2h}{\sqrt{b_i^{k+1}}+\sqrt{b_i^k}} \\ \text{subject to:} \quad & \\ & \mathbf{A}\mathbf{b}_i - 2h\,\mathbf{a}_i = 0, \mathbf{b}_i(1) = v_i^2(k_0), \\ & \mathbf{0} \le \mathbf{b}_i \le v_{\max}^2 \mathbf{1}, \quad a_{\min}\mathbf{1} \le \mathbf{a}_i \le a_{\max}\mathbf{1}, \\ & t_w(i) - [g({}^m\mathbf{b}_i) + \nabla g({}^m\mathbf{b}_i)^T(\mathbf{b}_i - {}^m\mathbf{b}_i)] \le 0. \end{aligned} \tag{$P({}^m\mathbf{b}_i)$}$$

Based on the idea of SCP, the optimization problem (9.7) can be resolved by solving $(P({}^m\mathbf{b}_i))$ iteratively. The details are given in Algorithm 13. At each iteration (Lines 3−10), the given point is set as the solution of the former iteration (Line 10). The iteration stops if it reaches the maximal iteration number (Line 2) or the error of two successive solutions/optimal values is less than the given precision (Line 7). Due to the approximation of the non-convex constraint (9.7f), $P({}^m\mathbf{b}_i)$ may have no optimal solution. However, our discretization guarantees that a robot can always stop at the end of a state if needed. This means, Problem (9.7) has feasible solutions at any time. Hence, in our approach, if (9.7) has no optimal solution, we would compute a feasible solution such that the robot stops at the end of its current state (Lines 11−13).

Algorithm 13: SCP procedure to solve (9.7).

Input : Current speed $v_i(k_0)$, waiting time $t_w(i)$, number of steps K, step length h, maximal number of iterations M, and precision ϵ.

Output: $\mathbf{b}_i$ and $\mathbf{a}_i$.

Initialization: $m = 0$, ${}^m\mathbf{b}_i = \mathbf{0}$, ${}^m obj_i = 0$;

while $m \leq M$ **do**

  Compute $g({}^m\mathbf{b}_i)$ and $\nabla g({}^m\mathbf{b}_i)$;

  Build the convex approximation $(P({}^m\mathbf{b}_i))$;

  Solve $(P({}^m\mathbf{b}_i))$;

  if there exists an optimal solution $\mathbf{b}_i$ and $\mathbf{a}_i$ **then**

    if $\|\boldsymbol{b}_i - {}^m\boldsymbol{b}_i\|_2 \leq \epsilon \,\|\, |obj_i - {}^m obj_i| \leq \epsilon$ **then**

      return $\mathbf{b}_i$ and $\mathbf{a}_i$;

    else

      $m = m + 1$; ${}^m obj_i = obj_i$; ${}^m\mathbf{b}_i = \mathbf{b}_i$;

  else

    `/* Compute a feasible solution so that` r_i `stops at the end of its current state. */`

    $\forall k \in \{k_0, \ldots, K-1\}$, $a_i^k = -v_i(k_0)^2/(2 * (K - k_0) * h)$, $b_i^{k+1} = b_i^k - v_i(k_0)^2/(K - k_0)$;

    return $\mathbf{b}_i$ and $\mathbf{a}_i$;

Algorithm 13 describes the optimal speed computation at step k_0 based on the detected environment at k_0. Once the acceleration is obtained, the robot can move along its path based on the kinematic equations (9.4). However, because of the change of other robots, the robot needs to update its speed real-timely based on the new hybrid states of other robots. Real-time speed control is realized via MPC. Detailedly, at each step, only the first element of the acceleration vector is applied; once it arrives at the next step, the robot updates its acceleration by recomputing the acceleration with the new hybrid states of other robots. Hence, the complete speed control at a discrete state is given in Algorithm 14. Lines 3–9 perform the discrete control and compute the waiting time: if the current transition is not enabled, compute waiting time based on enable-dependent robots (Lines 5–7); if the current transition is enabled but cannot fire, compute waiting time based on Algorithm 12 (Lines 8–9).

At last, we further discuss motion direction control along the given path in practice. Theoretically, since the path is determined, the motion direction at each position is predetermined. However, such motion direction usually oscillates greatly for curvic path

Algorithm 14: MPC-based speed control at state s.

Input : Physical constraints: $v_{\max}$, $a_{\max}$, and $a_{\min}$; current hybrid state at the start of s: $(s, x(0), v0, L)$; precision: ϵ.

/* Once r_i arrives at the start of s, do: */

Initialization: set proper step length h and number of steps K, $k_0 = 0$, $v_i(k_0) = v0$;

while $k_0 < K$ **do**

  $t_w(i) = 0$;

  Call Algorithm 11;

  if $decision = 0$ **then**

    Retrieve dependent robots $D_i(s)$;

    Compute waiting time $t_w(i)$;

  else if $decision = 1$ **then**

    /* Retrieve the sequence of waiting robots during the negotiation process. */

    $t_w(i) =$ Algorithm 12;

  Call Algorithm 13 and return $\mathbf{b}_i, \mathbf{a}_i$;

  $a_i^{k_0} = \mathbf{a}_i(1)$; /* select the first value. */

  $t[k_0 \to k_0 + 1] = \frac{\sqrt{\mathbf{b}_i(2)} - \sqrt{\mathbf{b}_i(1)}}{a_i[k_0]}$; /* compute the duration of the current step. */

  Move from L_{k_0} to L_{k_0+1} under (9.4) for a duration of $t[k_0 \to k_0 + 1]$;

  Update current step $k_0 = k_0 + 1$;

  Update its current hybrid state to $(s, x(k_0 h), \sqrt{\mathbf{b}_i(2)}, L - k_0 h)$

and cannot guarantee motion stability of a robot. In this work, due to its simplicity and efficiency, pure pursuit algorithm is applied to determine the motion direction. Once the acceleration at step k_0 is determined, the pure pursuit method is performed by regarding $x(L_{k_0})$ and $x(L_{k_0+1})$ as the start and the destination positions, respectively.

Take a 2D case as an example to introduce pure pursuit method briefly (refer to [133] for more details). As shown in Fig. 9.5, xOy is the body frame of the robot where y axis is the current motion orientation of the robot, the solid curve OAT is the predefined path, O is the current position $x(L_{k_0})$, T is the destination position (i.e., $x(L_{k_0+1})$ in our case), and A is a point on the path such that the distance between A and O is d_a, i.e., the look-ahead distance. Based on the geometry, we have $x^2 + y^2 = d_a^2$ and $(r - x)^2 + y^2 = r^2$. Hence, the curvature of the real motion path, i.e., the bold dashed curve in Fig. 9.5, can be written as: $\gamma = 1/r = 2x/d_a^2$. Let θ be the angle difference

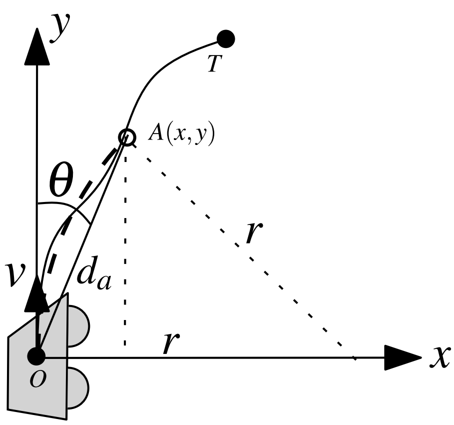


FIG. 9.5: An illustration of pure pursuit algorithm.

between the current orientation and that to point A. For a small difference, we have $\theta \approx \sin\theta = \frac{x}{d_a}$. Hence, the curvature can be re-formulated as $\gamma = \frac{2\theta}{d_a}$.

9.3.3 Effectiveness Analysis of the Proposed Approach

In this subsection, we give theoretical analysis of effectiveness of the proposed hybrid approach. Some notations are used in our analysis. For a robot r_i, its predicted uniform motion time is denoted as pt_i to move to the end of its current state with the current speed, and the real time to the end of its current state is rt_i. The optimal value of (9.7a) of r_i at the current position is $obj_i = obj_i(1) + obj_i(2)$, where $obj_i(1)$ and $obj_i(2)$ are the values of the first and second terms in (9.7a), respectively.

Lemma 8. If r_i can fire its current transition, then $pt_i \geq rt_i$.

Proof. If r_i can fire its current transition, then the time constraint (9.7f) does not need to be considered. In this situation, decelerated motion is not an optimal solution. Indeed, let obj_i^1 and obj_i^2 be the values under constant motion and under decelerated motion with respect to the current speed, respectively. First, it is clear that $obj_i^1(1) = 0$, while $obj_i^2(1) > 0$. Second, the motion time to a same position under a decelerated motion is larger than that under a constant motion, which means $obj_i^1(2) < obj_i^2(2)$. Thus, $obj_i^1 < obj_i^2$. This means that in this situation, r_i should move with its current speed or an accelerated speed. Hence, $pt_i \geq rt_i$. □

Based on the proof of Lemma 8, we can obtain the following lemma.

Lemma 9. If a robot can move at its current speed v_0, its optimal motion is either constant motion or accelerated motion with respect to v_0.

Lemma 10. The motion under the solution of problem (9.7) can guarantee collision and deadlock avoidance.

Proof. Suppose r_{i_0} at its current state s cannot transit to the next state, and the sequence of its enable-dependent robots is $D_i(s) = \{r_{i_1}, r_{i_2}, \ldots, r_{i_j}\}$, where r_{i_k} needs to wait for the move of $r_{i_{k+1}}$ for $k = 1, 2, \ldots, j-1$, and r_{i_j} can fire its current transition. The real motion time for r_{i_k} to the required position is denoted as rt_{i_k}. Based on Lemma 8, we have $pt_{i_j} \geq rt_{i_j}$. If $pt_{i_{j-1}} \geq pt_{i_j}$, $r_{i_{j-1}}$ can move at least with its current speed based on the proof of Lemma 9. Thus, we have $pt_{i_{j-1}} \geq rt_{i_{j-1}}$. Based on (9.7f), $rt_{i_{j-1}} \geq pt_{i_j}$. Hence, $pt_{i_{j-1}} \geq rt_{i_{j-1}} \geq pt_{i_j} \geq rt_{i_j}$ If $pt_{i_{j-1}} < pt_{i_j}$, $r_{i_{j-1}}$ needs to decelerate its motion based on its own optimal problem (9.7). In this case, $r_{i_{j-1}}$'s optimal solution should reach the boundary of its own constraint (9.7f). This means $rt_{i_{j-1}} = pt_{i_j} > rt_{i_j}$. Hence, the real motion time of $r_{i_{j-1}}$ satisfies $\max\{pt_{i_{j-1}}, pt_{i_j}\} \geq rt_{i_{j-1}} \geq rt_{i_j}$.

Suppose, $r_{i_{k+1}}$ satisfies $\max\{pt_{i_{k+1}}, \ldots, pt_{i_j}\} \geq rt_{i_{k+1}} \geq \ldots \geq rt_{i_j}$. Let us consider r_{i_k}. If $pt_{i_k} \geq \max\{pt_{i_{k+1}}, \ldots, pt_{i_j}\}$, r_{i_k} at least can move at its current speed based on its own (9.7). Thus, we have $pt_{i_k} \geq rt_{i_k} \geq \max\{pt_{i_{k+1}}, \ldots, pt_{i_j}\} \geq rt_{i_{k+1}}$. Note that in this case $pt_{i_k} = \max\{pt_{i_k}, pt_{i_{k+1}}, \ldots, pt_{i_j}\}$. If $pt_{i_k} < \max\{pt_{i_{k+1}}, \ldots, pt_{i_j}\}$, r_{i_k} needs to decelerate its motion. In this case, r_{i_k}'s optimal solution should reach the boundary of its own constraint (9.7f). This means $rt_{i_{j-1}} = \max\{pt_{i_{k+1}}, \ldots, pt_{i_j}\} > rt_{i_{k+1}}$. In conclusion, we have $\max\{pt_{i_k}, pt_{i_{k+1}}, \ldots, pt_{i_j}\} \geq rt_{i_k} \geq \ldots \geq rt_{i_{k+1}} \geq \ldots \geq rt_{i_j}$.

Based on the method of induction, we have that $rt_{i_0} \geq rt_{i_k} \geq \ldots \geq rt_{i_{k+1}} \geq \ldots \geq rt_{i_j}$. This means that when r_{i_0} arrives at the end of its current state, its enable-dependent robots have left their required positions. Hence, r_{i_0} can transit to its next state without any collisions and deadlocks. $\square$

Lemma 11. Algorithm 13 can always return a sub-optimal, if not optimal, solution of problem (9.7).

Proof. Based on our discretization, each robot is able to stop its motion from L_{k_0} to L_K. Thus, when $(P({}^m\mathbf{b}_i))$ fails to find an optimal solution, Lines 11–13 in Algorithm

13 guarantees to compute a feasible solution of (9.7). Otherwise, the optimal solution of $(P(^m\mathbf{b}_i))$ at $^m\mathbf{b}_i$ is a sub-optimal, if not optimal, solution of (9.7). Indeed, for a convex function $f(\mathbf{x})$, given an arbitrary point $\mathbf{x}_0 \in D$, we have $\forall \mathbf{x} \in D$, $f(\mathbf{x}) \geq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0)$. Since $g(\mathbf{b}_i) = \sum_{k=k_0}^{K-1} \frac{2h}{\sqrt{b_i^{k+1}}+\sqrt{b_i^k}}$ is a convex function, we have $g(\mathbf{b}_i) \geq g(^m\mathbf{b}_i) + \nabla g(^m\mathbf{b}_i)^T(\mathbf{b}_i - {}^m\mathbf{b}_i)$. Hence, $t_w - g(\mathbf{b}_i) \leq t_w - [g(^m\mathbf{b}_i) + \nabla g(^m\mathbf{b}_i)^T(\mathbf{b}_i - {}^m\mathbf{b}_i)]$. Compared with $(P(^m\mathbf{b}_i))$ and (9.7), the optimal solution of $(P(^m\mathbf{b}_i))$ is a feasible solution of (9.7). The sub-optimal can be derived directly from the local convergence of SCP [205]. □

Theorem 11. Under Algorithm 14, each robot can move under a sub-optimal, if not optimal, motion without causing collisions and deadlocks.

Proof. This theorem can be easily proved based on Lemmas 10 and 11 since Lemma 10 guarantees collision and deadlock avoidance at each time instant of the MPC procedure and Lemma 11 guarantees a sub-optimal, if not optimal, motion. □

9.4 Modeling of Communication Protocols in the Proposed Approach

As described in our approach, there does not exist a centralized controller for the system since each robot makes decision and moves independently by building and solving its own local optimization problem. To build its local optimization problem, each individual needs a sequence of communication with its neighbors in both discrete transition control and continuous speed adjustment. In this section, we discuss the communication protocols in our approach using Petri nets. A Petri net is a tuple $\langle P_P, T_P, A_P \rangle$, where P_P is a set of places, T_P is a set of transitions, and $A_P \subset ((P_P \times T_P) \cup (T_P \times P_P))$ is a set of arcs.

Before giving the detailed Petri net models, we summarize the messages used in the protocols.

TABLE 9.1: Messages for Communication Among Robots

Class	Formula	Meaning of Message Content
msg_1	$\langle snd, rec, (r, s)\rangle$	Robot r activates the communication procedure for $Dect(r, s)$.
msg_2	$\langle snd, rec, (r, idle)\rangle$	Notification to r that the detection of a robot whose succeeding state is idle but is not s.
msg_3	$\langle snd, r, (snd, s)\rangle$	snd asserts the detection of s.
msg_4	$\langle snd, rec, (s)\rangle$	rec should move away from s before snd arrives at s.
msg_5	$\langle snd, rec, (Info)\rangle$	$Info = \{(r_j, v, L)\}$, where r_j is a waiting-for robot, v is its current speed, and L is the path required motion length.

Definition 38. A message transmitted between two robots is a tuple $\langle snd, rec, (c)\rangle$, where snd is the robot sending the message, rec is the one receiving the message, and (c) is the content of the message.

In our approach, there are two communication protocols: one is for deadlock detection and the other is to retrieve the waiting-for robots. A waiting-for robot is a robot that another one needs to wait for. Based on the content of messages, messages transmitted among robots are divided into five classes, whose details are described in Table 9.1. The first three are for deadlock detection and the last two are for retrieval of waiting-for robots. In the sequel, taking r_i as an example, we describe the protocols in details.

First, consider the communication protocol for r_i's deadlock detection $Dect(r_i, s)$. Fig. 9.6 shows the Petri net model of an intermediate robot involved in $Dect(r_i, s)$. In this model, B_1^1 and B_2^1 are communication buffers for msg_1 in the communication network, B_1^2 and B_2^2 for msg_2, and B_1^3 for msg_3. P_1: prepare for communication; T_1: receive a msg_1 message whose content is (r_i, s) and check the status of the succeeding state; P_2: the status of the checked state; c1 $-$ c3: three conditions specifying the status stored in P_2, where c1 is "the checked state is not occupied by any robot and does not match s", c2 is "the checked state is occupied by another robot", and c3 is "the checked state is s". T_2: send a msg_3 message to assert the detection of s; T_3: publish a msg_2 message to confirm the idleness of its succeeding state; T_4: publish a msg_1 message to the robot at its succeeding state; T_5: publish the received msg_2 message.

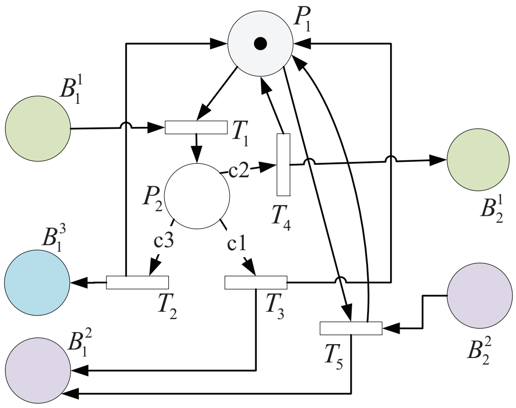


FIG. 9.6: Communication model of an intermediate robot involved in $Dect(r_i, s)$.

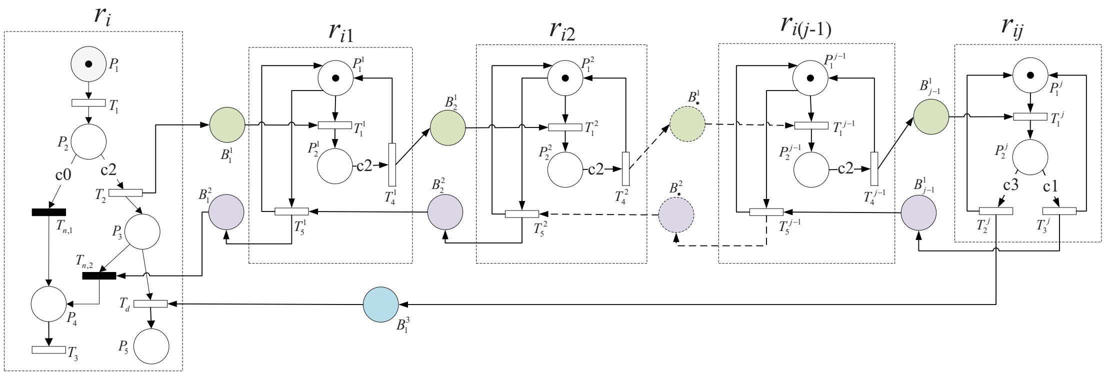


FIG. 9.7: Communication protocol for the deadlock detection procedure $Dect(r_i, s)$.

Take r_{ik} as an example to explain the evolution of the net. When r_{ik} receives a msg_1 message from B_1^1, say $\langle r_{i(k-1)}, r_{ik}, (r_i, s)\rangle$, T_1 is enabled and fires to check the succeeding state. If c1 satisfies, T_3 fires and a msg_2 message $\langle r_{ik}, r_{i(k-1)}, (r_i, idle)\rangle$ is published to buffer B_1^2. If c3 satisfies, T_2 fires and a msg_3 message $\langle r_{ik}, r_i, (r_{ik}, s)\rangle$ is published to buffer B_1^3. If c2 satisfies and the checked state is occupied by $r_{i(k+1)}$, then T_4 fires and a msg_1 message $\langle r_{ik}, r_{i(k+1)}, (r_i, s)\rangle$ is published to buffer B_2^1. Note that after firing each of $T_2 - T_4$, r_{ik} also stores $r_{i(k-1)}$ to P_1 for a given duration. When it receives a msg_2 message $\langle r_{i(k+1)}, r_{ik}, (r_i, idle)\rangle$ from B_2^2, T_5 fires and publishes a msg_2 message $\langle r_{ik}, r_{i(k-1)}, (r_i, idle)\rangle$ to B_1^2 by checking the stored information $r_{i(k-1)}$.

Fig. 9.7 shows the first communication protocol of r_i for the procedure $Dect(r_i, s)$. Explanations of some places and transitions are given as follows. P_1: initialize $Dect(r_i, s)$; T_1: check the status of $Pos_i(s)$; c0: $Pos_i(s)$ is a private state or is idle; $T_{n,1}$ and $T_{n,2}$: confirm no deadlocks; T_2: publish a msg_1 message to the next robot; T_d: confirm a deadlock with the received response; P_3: wait for response; P_4: no deadlocks;

P_5: initialize the communication process for retrieval of waiting-for robots; and T_3: finish the communication for deadlock detection. For an internal robot, i.e., $r_{i1}, \ldots, r_{i(j-1)}$, only c2 is satisfied and its T_4 fires; while the terminal robot r_{ij} can satisfy either c1 or c3, resulting in the firing of T_2^j or T_3^j, respectively.

If r_i needs to do deadlock detection, T_1 fires to check $Pos_i(s)$, whose status is either idle or occupied by another robot. If $Pos_i(s)$ is idle, then confirmation of no deadlocks is generated by firing $T_{n,1}$. However, if r_i detects that $Pos_i(s)$ is occupied by another robot, say r_{i1}, r_i sends a message $\langle r_i, r_{i1}, (r_i, s)\rangle$ to r_{i1} by firing T_2. When r_{i1} receives this message, it checks its succeeding state (firing T_1^1) and detects r_{i2}, then a msg_1 message $\langle r_{i1}, r_{i2}, (r_i, s)\rangle$ is sent to r_{i2} (firing T_4^1). With the model described in Fig. 9.7, the content (r_i, s) is transmitted one by one among the internal robots until there exists a robot r_{ij} which checks its succeeding state satisfying c1 or c3. If c3 is satisfied, i.e., the succeeding state is s, then a msg_3 message $\langle r_{ij}, r_i, (r_{ij}, s)\rangle$ is sent to r_i directly since their distance cannot be very large. After receiving this message, T_d fires and r_i is initializing the communication for waiting-for robot retrieval. If c1 is satisfied, i.e., the succeeding state is idle and is not s, a msg_2 message is first transmitted to $r_{i(j-1)}$, then one by one to r_{i2} and r_{i1}. At last, r_i receives this message content from r_{i1} and asserts that no deadlocks are detected by firing $T_{n,2}$. Since we cannot guarantee that r_{ij} is still in the communication range of r_i for this situation, and we can only guarantee that it is in the communication range of $r_{i(j-1)}$. Thus, the msg_2 message is transmitted to r_i via a sequence of internal robots.

Second, consider the communication protocol for the retrieval of waiting-for robots. Note that the process is that: when r_i detects a collision or deadlock if it was at a state s, r_i should wait for a robot, say r_j, to move away from s first. Thus, r_j should check whether it can move to its succeeding state $Pos_j(s)$. If r_j also needs to wait for a robot r_{j_1}, then r_{j_1} checks whether it can move to a required state. Continue the process until a robot can move to a given state. Fig. 9.8 describes the communication model of robots, e.g., r_j and r_{j_1}, involved in r_i's retrieval of waiting-for robots. In the model, B_1^4 and B_2^4 are buffers for msg_4 message, and B_1^5 and B_2^5 are buffers for msg_5 message. Suppose the received message from B_1^4 is $\langle r_i, r_j, (s)\rangle$. T_1: check the status of $Pos_j(s)$; c4: the checked state is idle; c5: the checked state is occupied by another robot; T_2: start

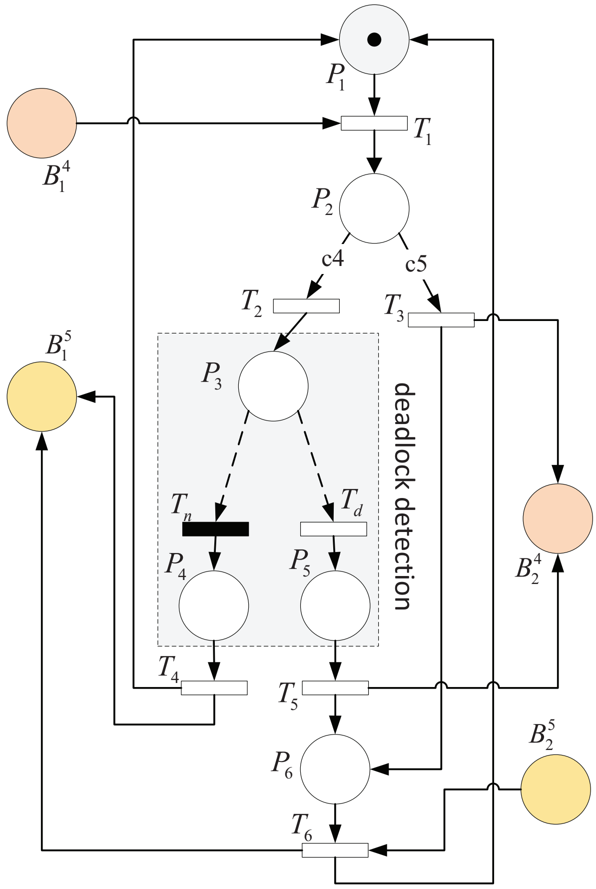


FIG. 9.8: Communication model of an intermediate robot involved in r_i's procedure to retrieve its waiting-for robots.

deadlock detection $Dect(r_j, Pos_j(s))$; P_3: initialize $Dect(r_j, Pos_j(s))$; T_n: confirm no deadlocks; T_4: send to B_1^5 its current speed and the path length required to move; T_d: confirm a deadlock; T_3: publish a msg_4 message to B_2^4 in order to avoid collisions; T_5: publish a msg_4 message to B_2^4 to inform the terminal robot of $Dect(r_j, Pos_j(s))$ to move away from $Pos_j(s)$; P_6: wait for the response from its waiting-for robot; T_6: update the content of received msg_5 message from B_2^5 and publish to the buffer B_1^5.

Detailedly, when it receives $\langle r_i, r_j, (s)\rangle$ from buffer B_1^4, r_j needs to check whether it can move to $Pos_j(s)$. So r_j checks the status of $Pos_j(s)$ by firing T_1. If $Pos_j(s)$ is occupied by a robot, say r_k, then c5 is satisfied. Hence, T_3 fires and publishes a msg_4 message $\langle r_j, r_k, (Pos_j(s))\rangle$ to B_2^4, and r_j is waiting for the response from r_k. Otherwise, c4 is satisfied and T_2 fires to start $Dect(r_j, Pos_j(s))$. The shaded part is the procedure to detect deadlock described in Fig. 9.7. It results in either no deadlocks, i.e., T_n can fire, or a deadlock with terminal robot, say r_k, i.e., T_d fires. In the former

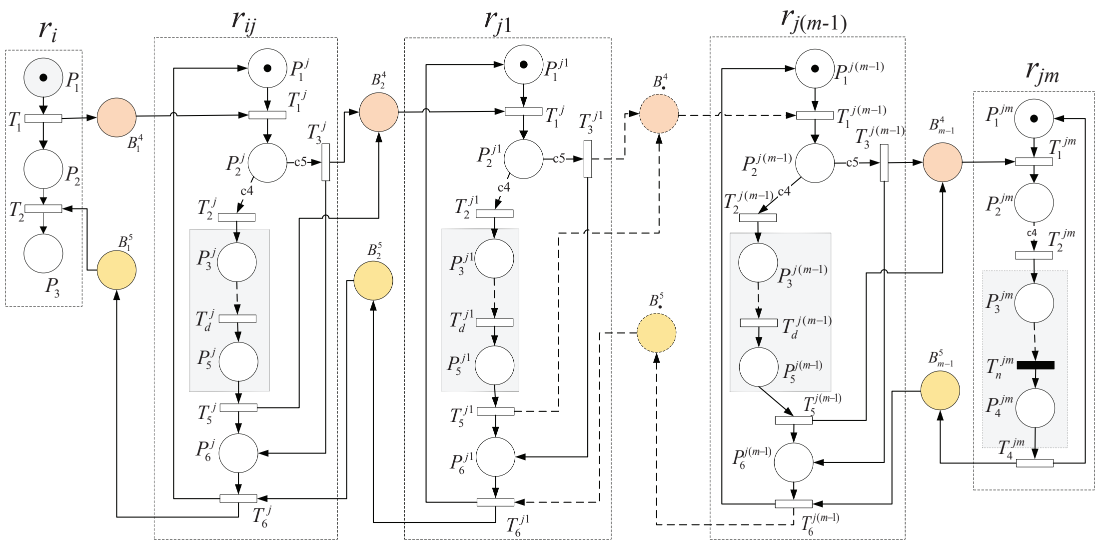


FIG. 9.9: Communication protocol of r_i for its procedure to retrieve waiting-for robots.

case, r_j fires T_4 and publishes a msg_5 message $\langle r_j, r_i, Info\rangle$ to B_2^5, where $Info = \{(r_j, v_j, L_j)\}$ and L_j is the path length from r_j's current position to the end of s. In the latter case, T_5 fires, publishing a msg_4 message $\langle r_i, r_k, (Pos_j(s))\rangle$ to B_2^4, and r_i is waiting for response from r_k. When it receives $\langle r_k, r_j, Info\rangle$ from B_2^5, T_6 fires and publishes to B_1^5 a new msg_5 message $\langle r_j, r_i, Info'\rangle$ whose content is updated with $Info' = Info \cup \{(r_j, v_j, L_j)\}$.

Fig. 9.9 shows the second communication protocol of r_i to retrieve its waiting-for robots when r_i detects a collision or deadlock and needs to wait for r_{ij} to move away from s. P_1 denotes the initialization of the retrieval communication with the information (r_{ij}, s). Hence, r_i starts to retrieve its waiting-for robots by firing T_1 and sends a msg_4 message $\langle r_i, r_{ij}, (s)\rangle$ to r_{ij}. Then, based on the model shown in Fig. 9.8, each internal robot r_k, $r_k \in \{r_{ij}, r_{j1}, \ldots, r_{j(m-1)}\}$, receives a msg_4 message, say $\langle r_{k_1}, r_k, (s_{k_1})\rangle$, from r_{k_1} and sends a new msg_4 message $\langle r_k, r_{k_2}, (Pos_k(s_{k_1}))\rangle$ to the next robot r_{k_2}. The communication returns back by a terminal robot, e.g., r_{jm}, which finds that it can move to the required state $Pos_{jm}(s_{j(m-1)})$ when it receives $\langle r_{j(m-1)}, r_{jm}, s_{j(m-1)}\rangle$ from B_{m-1}^4. Then, T_4^{jm} fires and sends back a msg_5 message $\langle r_{jm}, r_{j(m-1)}, Info\rangle$, where $Info = \{(r_{jm}, v_{jm}, L_{jm})\}$ and L_{jm} is the path length from r_{jm}'s current position to the end of $s_{j(m-1)}$. The content of this kind of message is updated by the internal robots one by one. Finally, r_i receives the msg_5 message, retrieves its waiting-for robots as well as related information, and completes the communication by firing T_2.

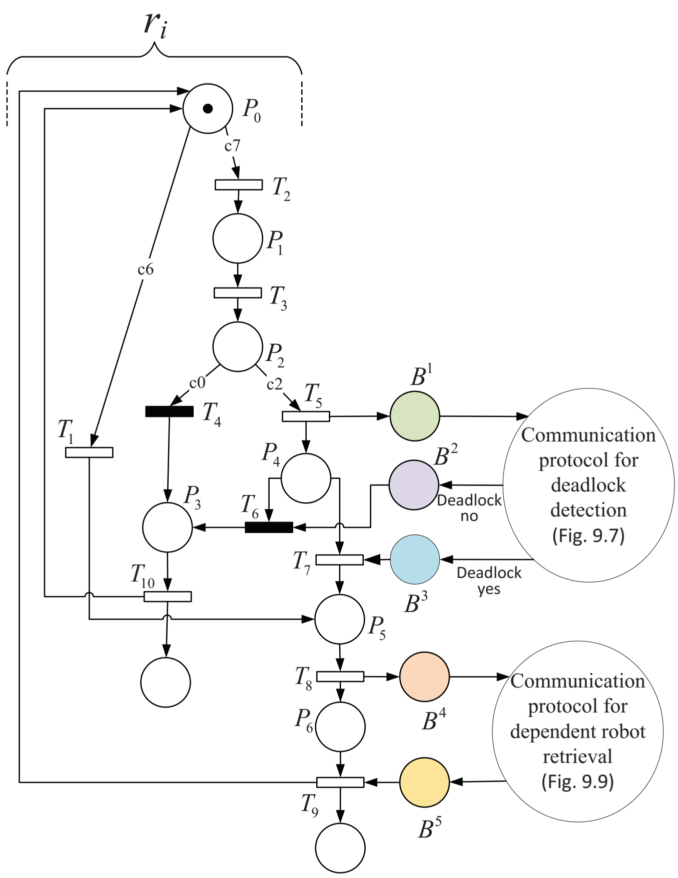


FIG. 9.10: Communication architecture of robot r_i.

Based on the protocols described in Figs. 9.7 and 9.9, we can obtain the complete communication architecture for r_i, which is shown in Fig. 9.10. Suppose r_i's current state is $s_{cur,i}$. Let $s = Pos_i(s_{cur,i})$, and $ss = Pos_i(s)$. Note that the communication can be launched only when s is a collision state. Thus, in Fig. 9.10, P_0: prepare for communication; c6: s is occupied by another robot; c7: s is idle; T_1: launch the communication for retrieval of waiting-for robots; T_2: launch the process $Dect(r_i, s)$; T_{10}: finish the communication process and return the communication result; and others have the same meanings as those in Figs. 9.7 and 9.9. If s is occupied by another robot, i.e., c6 is satisfied, this means that a collision is detected. So T_1 is fired and r_i is at P_5: initialization of the second kind of communication, i.e., communication process for retrieval of waiting-for robots. If s is idle, i.e., c7 is satisfied, this means that r_i should perform deadlock detection. So T_2 is fired, and r_i reaches the status of initialization of the communication for deadlock detection, i.e., P_1. Hence, r_i launches the communication for deadlock detection and then that for waiting-for robot retrieval.

Based on the above protocols, a robot can communicate with its neighbors, determine whether its motion will cause collisions or deadlocks, and finally compute its minimal motion time at the current state, based on which it can compute its speed independently. In this way, the motion controller of each robot can control its motion in a distributed manner. Thus, we have the following proposition.

Proposition 8. The multi-robot system under the proposed control approach is a distributed control system.

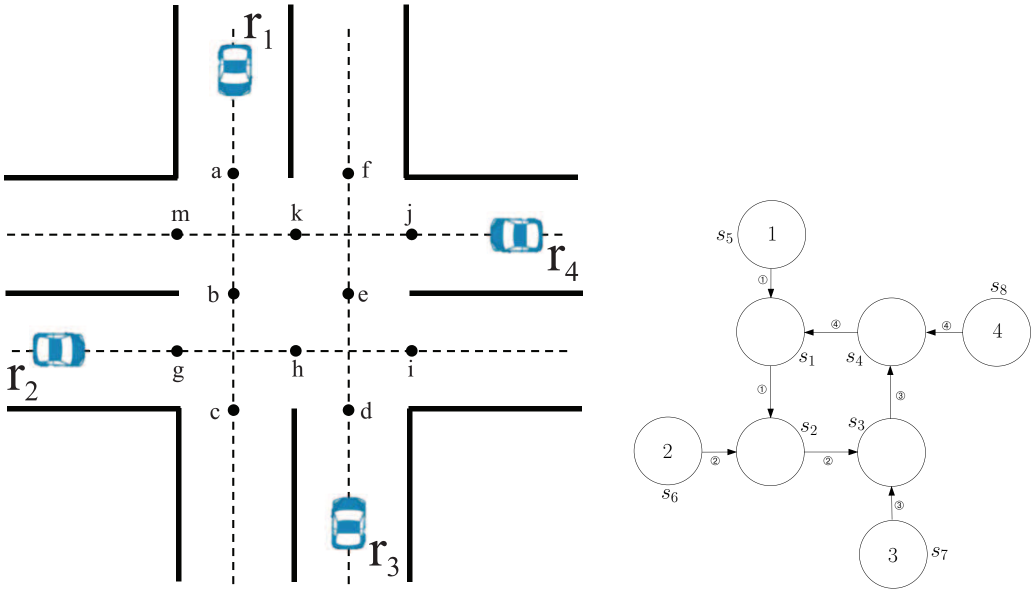
9.5 Simulation Cases

In this section, we give some simulations to show the effectiveness of our approach. Our experiments are done via MATLAB. The toolbox CVX [199] with MOSEK solver is applied to implement SCPs.

9.5.1 Simulation Results under the Proposed Hybrid Approach

First, suppose there are four autonomous vehicles, r_1, r_2, r_3 and r_4, moving through a cross shown in Fig. 9.11(a), where the position of a vehicle is described as its center of mass. The distance from their current positions to the cross boundaries a, g, d, and j are equal to 3ρ, and the length of each segment between any two successive points is equal to 4ρ, where $\rho = 100$ units. The corresponding transition system of each robot is shown in Fig. 9.11(b). Detailedly, the segments from their current positions to a, g, d, and j are abstracted to private states s_5, s_6, s_7, and s_8, respectively; segments ab and mk, are abstracted to a collision state s_1, bc and gh as s_2, hi and de as s_3, and ef and jk as s_4.

The parameters of robots are as follows: speed constraint of each robot is $v \in [0, 100]$, acceleration constraint is $a \in [-150, 150]$, the step length of each robot is set as $h = 100/3$, and the numbers of steps of at each private and collision states are 9 and 12, respectively.



(a) Path network. (b) Transition system.

FIG. 9.11: Paths of four robots and the corresponding transition system.

In the simulation, we assume r_1, r_2, r_3, and r_4 arrive at the starts of s_5, s_6, s_7, and s_8 with speed 60, 50, 40, and 30, respectively. The evolutions of acceleration, speed and distance during the simulation are shown in Figs. 9.12, 9.13, and 9.14. Since they are moving to a collision region $X = \{s_1, s_2, s_3, s_4\}$, these robots need negotiation. Based on Algorithm 10, r_1, r_2, and r_3 can fire their current transitions, while r_4 needs to wait for r_3 to move away from s_4. Thus, at the beginning, r_4 has negative acceleration (the dotted line in Fig. 9.12) to slow down its speed (the dotted line in Fig. 9.13), while others keep their current speeds.

At time instant t_1, r_1 transits to s_1. Then the negotiation process determines that r_1 needs to wait for r_2 to move away from s_2 and r_4 still needs to wait for r_3. Thus, r_1 slows down its motion. This can be seen from both its acceleration in Fig. 9.12 and speed in Fig. 9.13 (the dash-dot lines). At time instant t_2, r_2 arrives at s_2. Since r_3 will transit to s_3 earlier than r_2, r_2 needs to wait for r_3. Thus, r_2 also decreases its speed. At time instant t_3, as shown in Fig. 9.14, r_1, r_2, and r_3 arrive at the end of s_1, s_2, and s_3 at the same time. Thus, when r_3 transits to s_4, r_2 transits to s_3 and r_1 to s_2. At this time, based on the optimal objective, r_2 keeps a constant speed and r_1 first accelerates its motion and then keeps a constant speed. Note that r_4 still decelerates its motion in order to avoid collision with r_3. At time instant t_4, r_3 leaves away from

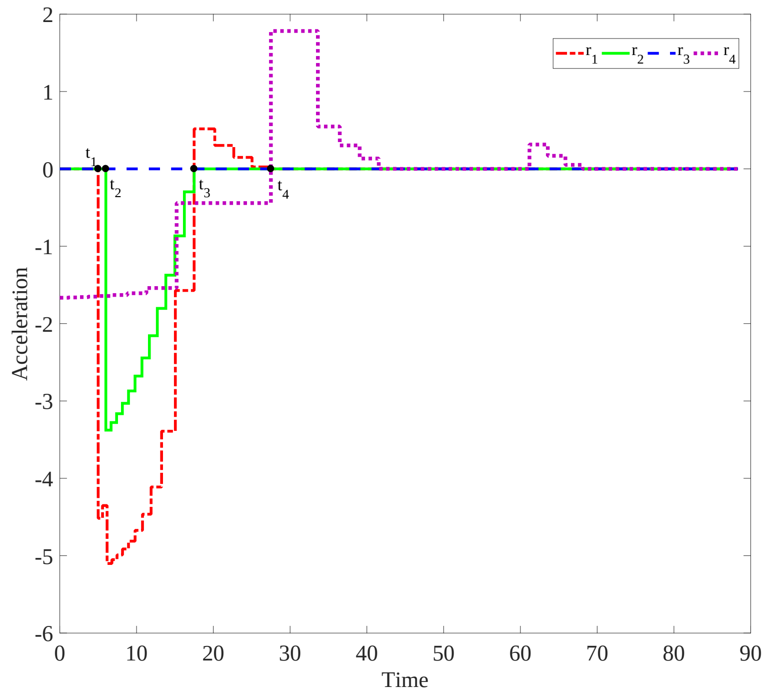


FIG. 9.12: Acceleration of the four robots in the simulation.

s_4 and r_4 transits to s_4. Since it does not need to wait for any robots, after t_4, r_4 first accelerates its motion and then keeps a constant speed. The simulation video can be found at `https://youtu.be/C21fDFU4Nyo`.

From the change of acceleration shown in Fig. 9.12 and speed in Fig. 9.13, we can conclude that during the motion along the cross, each robot adjusts its speed to avoid collisions and deadlocks, as well as keeps its motion as smoothly as possible.

9.5.2 Comparison of Our Approach with Discrete Control

In this subsection, we compare the motion under our hybrid control and that under only discrete control. Since there is no speed controller considered in discrete control, a natural way to drive a robot is that it first moves in a constant speed; if it finds that it cannot transit to the next state, the robot will decelerate its motion with the largest deceleration near the end of the current state so that it can stop its motion at the end; when it can move forward again, the robot accelerates to its previous speed. Besides, the first robot reaching a state can transit to this state first as long as the transition cannot

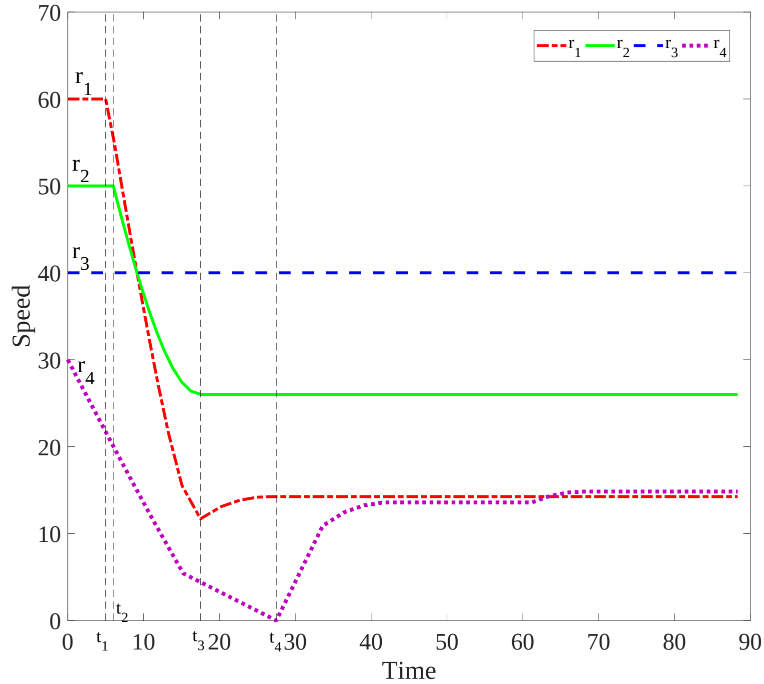


FIG. 9.13: Speed of the four robots in the simulation.

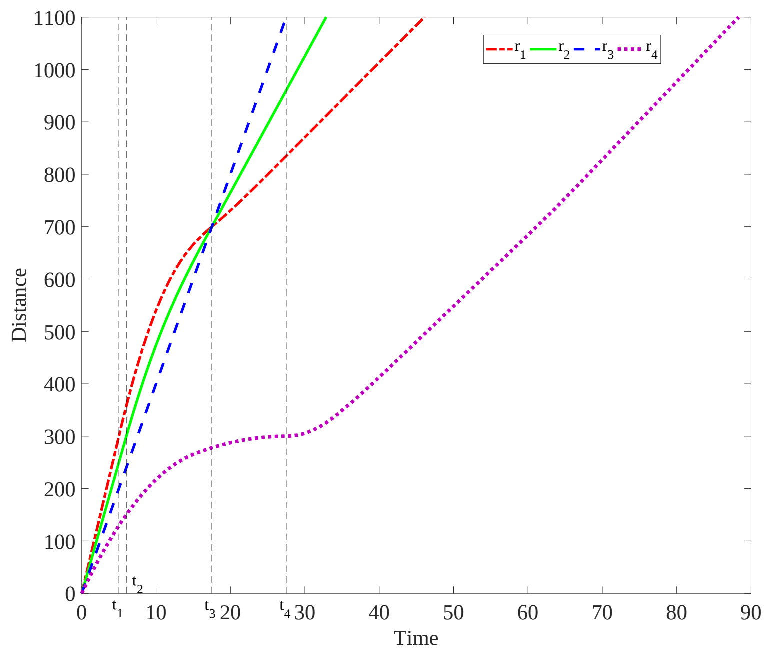


FIG. 9.14: Distances of the four robots in the simulation.

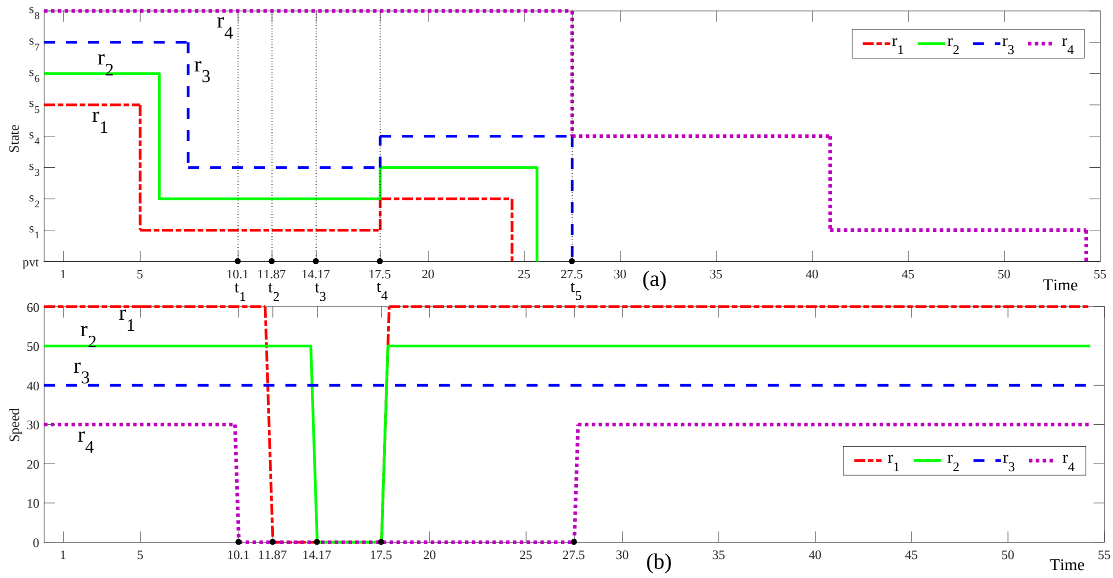


FIG. 9.15: Simulation results with only discrete control.

cause any collision and deadlock. Based on this idea, we can give the simulation results of the system described in Fig. 9.11 under discrete control.

Fig. 9.15 shows the state transition and speed of the four robots under discrete control, where pvt means a private state of a robot. As shown in Fig. 9.15(a), before r_4 reaches the end of s_8, r_1, r_2, and r_3 are at s_1, s_2, and s_3, respectively. To avoid a deadlock, r_4 stops its motion with the maximal deceleration such that it can stop at the end of s_8. Hence, as shown in Fig. 9.15(b), r_4 stops its motion at time instant t_1, while r_1, r_2, and r_3 are still at s_1, s_2, and s_3, respectively. Near the time instant t_2, r_1 predicts that when it arrives at the end of s_1, r_2 is still at s_2. Thus, r_1 stops its motion at time instant t_2, when r_1 reaches the end of s_1, as shown in Fig. 9.15(b). Similarly, before reaching the end of s_2, r_2 predicts that r_3 is still at s_3. So r_2 stops its motion at the end of s_2 with its maximal deceleration at time instant t_3. At time instant t_4, r_3 transits to s_4, so r_2 can move forward and then r_1 moves forward, but r_4 should still stop at s_8. Hence, r_1 and r_2 resume their motion with their maximal acceleration to their former speed and then move with constant speed. At t_5, r_3 moves away from s_4, so r_4 resumes its motion with its maximal acceleration and then keeps a constant speed.

Comparing the speeds in Fig. 9.13 and Fig. 9.15(b), we can find that: (1) there are much less stops under the proposed hybrid approach than pure discrete control; (2) whenever a robot needs acceleration or deceleration, there is no sharp change of the

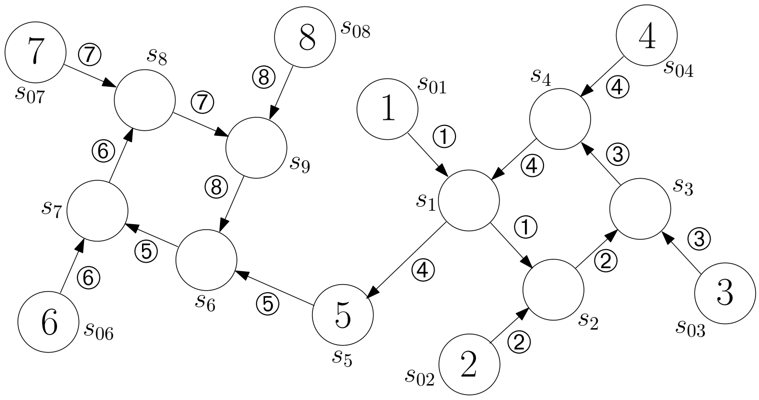


FIG. 9.16: A more complex simulation example.

speed under hybrid approach. Hence, the proposed approach can guarantee smooth motion of robots. Indeed, when a robot detects a deadlock if it arrives at its next state, the robot is required to stay at its current state for a proper time duration in order to avoid the detected deadlock. In the pure discrete method, each robot keeps a constant speed during the motion at the state, which may lead to a shorter motion time at this state than the required time, and thus it must decelerate immediately to stop at the end of the related segment. While in our proposed method, with the continuous control part, a robot adjusts its speed based on the waiting time such that the robot can reach the end of the current segment with an intelligently-tuned smooth speed change. In this way, our approach leads to the advantages of fewer stops and fewer sharp-speed changes.

9.5.3 A More Complex Scenario

To further show the effectiveness of our approach, we study a more complex transition system, which is shown in Fig. 9.16, where the states with numbers denote the current states of the corresponding robots. The parameters of the robots are the same with those of the system studied in Section 9.5.1, but with different speeds. Suppose the eight robots, $r_1, \ldots, r_8$, arrive at the starts of their current states with speeds 35, 65, 55, 45, 30, 60, 50, 40, respectively.

The task of the robots in this experiment is to pass through the collision region $X = \{s_1, s_2, \ldots, s_9\}$. The evolutions of their speed and state transition during the simulation are shown in Figs. 9.17 and 9.18. The vertical lines denote the time instants

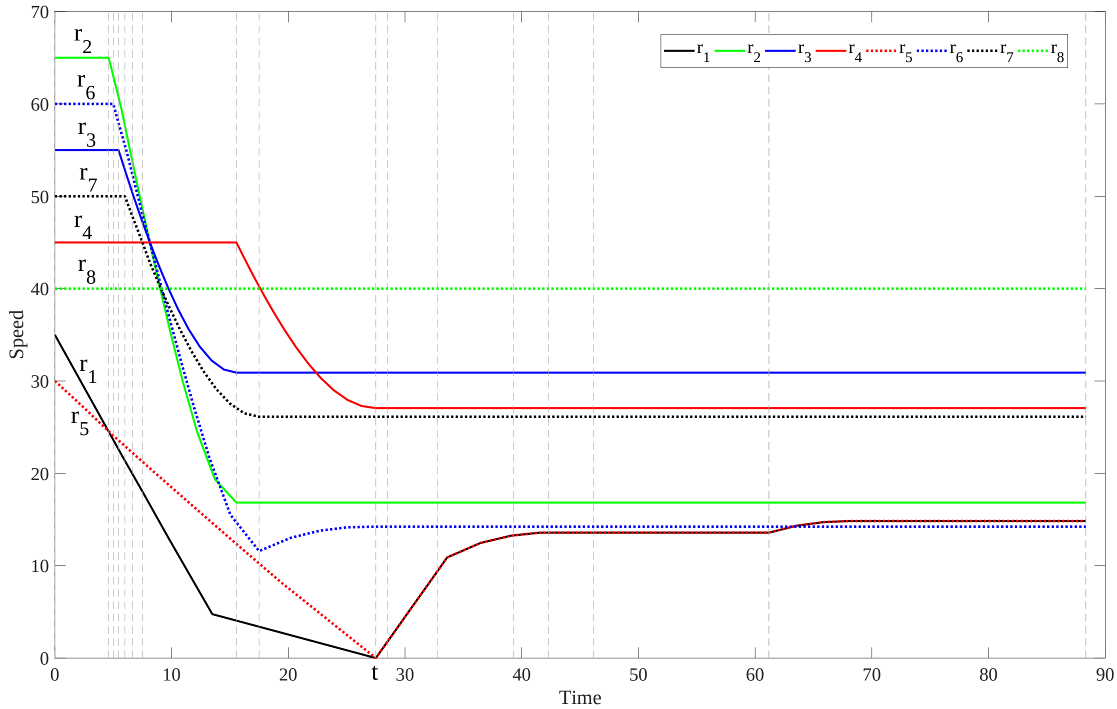


FIG. 9.17: Speed evolution of the robots.

when some robots fire their transitions. First, r_1 needs to wait for r_4 and r_6 needs to wait for r_8 during the negotiation process. So the two robots decrease their speed, as shown in Fig. 9.17. As shown in Fig. 9.18, r_2 transits to s_2 at the time instant denoted by the first vertical line. At this instant, r_2 should decelerate its motion based on the negotiation since r_3 will arrive at s_3 earlier. Similarly, when r_6 moves to s_7, it should decrease its speed since another robot, r_7, will arrive at s_8 earlier than r_6. So do r_3, r_7, and r_4. Note that, as shown in Fig. 9.18, at time instant t, r_4 transits to s_5 and r_8 moves to its own private state. So r_1 and r_5 transit to s_1 and s_6, respectively. Thus, the current configuration of the system is $(s_1, s_3, s_4, s_5, s_6, s_8, s_9, pvt)$. Clearly, at this configuration, each robot can move without any waiting. Since the speed of r_1 and r_5 are 0, and their parameters are the same, their optimal solutions are the same. Hence, as shown in Fig. 9.17, their speeds are the same along with time.

9.6 Conclusion

In this chapter, we study a distributed and hybrid approach to motion control of a multi-robot system where each robot has a predefined path. Our approach contains two phases.

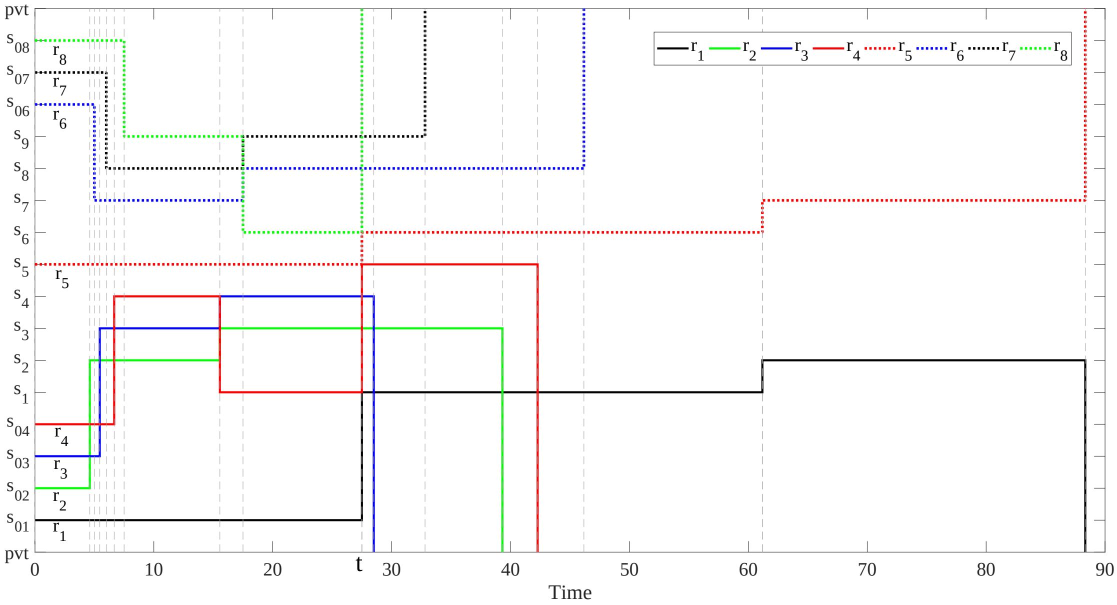


FIG. 9.18: State transitions of the robots.

At the first phase, an online discrete control policy is proposed to determine whether a robot can transit to the next state in order to avoid collisions and deadlocks. Based on the discrete determination, at the second phase, an MPC-based continuous control strategy is proposed to compute the optimal speed of a robot such that it can obey the given decision. Each optimization problem constructed at this phase only contains a robot's physical constraints and one time related constraint. This reduces the scale of the optimization problem greatly.

Chapter 10

Conclusion and Future Research

In this chapter, we first give a summary of the work conducted in this thesis and then discuss some future research directions based on our current results.

10.1 Summary

Motion control for multi-robot systems is one of the most important issues. On one hand, as an individual, each robot needs to avoid collisions and deadlocks with others; on the other hand, as a whole system, all robots in a system need to cooperate with others during their motion. To leverage the advantages of multi-robot systems, in this thesis, we concentrate on distributed approaches to motion control of robots in multi-robot systems, which not only guarantee good flexibility of the systems but also enhance communication among robots.

First, we study a fully distributed and real-time trajectory planning method. This approach applies MPC to realize time receding so as to update environment parameters and communication connection, and iSCP to resolve each robot's local optimization problem on each horizon. In our approach, a robot only needs to (1) detect environmental obstacles in the current sensing range and (2) communicate with robots within its communication range to retrieve their current positions and velocities, which can be obtained immediately. By predicting its neighbors' motion as uniform attributes based on the retrieved information, a robot can finally build its optimization problem to avoid collisions and compute its motion input, i.e., acceleration, to its actuator.

Second, when the paths are recorded from the above work, the future robots may be required to move along these paths in some scenarios. Or a robot has to move along a predefined path due to some infrastructure limitations. In such systems, since the paths of robots are determined, robots may cause not only collisions but also deadlocks. Hence, in the sequel, we propose a distributed approach to avoiding collisions and deadlocks in multi-robot systems with predefined path networks. We first model the motion of robots in such a system by LTS models. Based on its LTS model, a robot checks its succeeding state to determine whether there exists a collision. To avoid deadlocks, a robot needs to check its next two states and communicate with other robots via a multi-hop communication path to detect deadlocks. Only if its current move transition does not cause any collisions or deadlocks can a robot move one step forward.

Third, for some complex path networks, avoiding a deadlock may cause other circular waits, which also make robots stop their motion. Recursively, even though no deadlocks may occur after the firing of its current transition, a robot also cannot move forward. Hence, we further study the avoidance of higher-order deadlocks, which are deadlock-free configurations currently but under which deadlocks will occur inevitably. We investigate the structural properties of higher-order deadlocks and propose a distributed approach to avoiding higher-order deadlocks. A system with N robots can at most form a higher-order deadlock of order $N-3$, meaning that a deadlock will occur within $N-3$ steps. Hence, each time a robot only needs to check at most $N-1$ states. To detect a higher-order deadlock, a robot should communicate with other robots via a multi-hop communication path to determine whether there exists any circuit and whether a circuit may be a higher-order deadlock.

Fourth, we study motion control for systems containing reliable and unreliable robots. We study robust control in such systems, whose target is to minimize the effect of failed robots on a system. We focus on systems with fixed paths since for the systems which can change paths, robust control can be achieved by regarding the failed robots as obstacles. Based on the LTS models, we propose a distributed approach, which contains two kinds of local algorithms: one for reliable robots and the other for unreliable ones.

It mandates that the failure of a robot can only affect the motion of robots that collide directly with the failed ones.

At last, we study a distributed, hybrid, and real-time motion control method of a system with fixed paths. It combines both continuous and discrete technologies studied before. Based on MPC strategy, on each horizon, with discrete control, a robot can determine the robots it needs to wait for in order to avoid collisions and deadlocks; then it predicts the least motion time it should spend in its current state; based on this information, the robot can build its local optimization problem and solve it using SCP; finally, the first acceleration is applied to control the robot, and the process is repeated on the next horizon. The communication protocols are also implemented via Petri nets. With the proposed hybrid approach, a robot not only can avoid collisions and deadlocks efficiently, but also can obtain the low-level continuous inputs to its actuator.

10.2 Future Work

This thesis proposes some distributed algorithms for motion control in multi-robot systems. In the future, there are some interesting research directions that can be further investigated.

1. Implementation of the Developed Algorithms on Real Robots. This thesis focuses on the design of novel motion planning and control algorithms and their theoretical analysis. Hence, simulation results are enough to show their effectiveness and efficiency. In the future, we will implement the developed algorithms on our real robot platforms, i.e., Asctec Hummingbird UAVs, Asctec Pelican UAVs, and Toyota COMS AV, and demonstrate the performance in real world environments, which increase the potential impact of the work.

2. Motion Control in Complex Scenarios. Currently, we only focus on the requirement that each robot can move to its given target position. In the future, we will further consider more complicated tasks. For example, the robots in a system should not only arrive at their target positions successfully but also maintain some specific formations

as accurately as possible during their motion; some robots have predetermined priorities to pass through some areas due to different importance of their assigned tasks. Another topic can be on more complex paths for each robot. For example, a robot may have multiple paths to move along, or a path may also contain multiple robots; in these scenarios, collisions and deadlocks are more sophisticated, and we should determine whether there exist any higher-order deadlocks, and investigate their properties and avoidance if higher-order deadlocks do exist.

3. Performance Optimization in Controlled Systems. On the basis that each robot can complete its tasks, we would further optimize the performance of the controlled multi-robot systems. First, rather than consider only one objective, we may further consider multi-objective optimization, such as minimizing the total motion time and the energy cost, and maximizing the motion stability. Game theory [206, 207] then may be a powerful tool to deal with motion planning with multiple objectives in multi-robot systems. Second, we may refine the discrete model of robot motion to allow more admissible behavior. For example, as shown in Fig. 5.2, the collision pair $(\widehat{agbhc}, \widehat{dgehf})$ can be divided into two pairs $(\widehat{agb}, \widehat{dge})$ and $(\widehat{bhc}, \widehat{ehf})$, either of which is abstracted as a collision state; in this way, there may exist two states connected with multiple arcs. Third, instead of deterministic models, we can apply probabilistic models to describe robot failures and study the related distributed robust control algorithms; we may also study how to evaluate failures and reliability of robots [208, 209]. At last, for distributed approaches, we can also study optimization of the negotiation process among robots. Some game theoretic approaches [210–212] can be used as reference.

4. Resilience to Attacks via Logic Control. Besides the topic of motion planning and control, with the development of technologies in robotics, security problems become more and more important for robots. On one hand, a robot needs some mitigation measures to protect itself from failures and attacks. For example, self-adaptive systems [213, 214] are well-designed technologies for robots to change their behavior when facing failures or attacks. On the other hand, since the fundamental behavior of a robot is to move from one position to another physically, monitoring and mitigating attacks from motion control logics, i.e., designing motion control algorithms resilient against attacks, is also an important and attractive topic. Besides, if a robot is finally attacked,

it may become an adversary in a multi-robot system, and thus we should study motion control in a multi-robot system with adverse robots.

5. Machine Learning in Robot Motion Control. Last but not the least, we will apply machine learning technologies, such as deep reinforcement learning, to aid the above research topics, from motion control to attack resilience. First of all, we will collect a large amount of data by running extensive real-world experiments, so that we can train high-quality machine learning models, which can be used to aid motion control, defend against adversaries, etc. Second, we can study how machine learning can help facilitate motion planning and control. For example, unlike uniform motion prediction, we may use machine learning models, such as deep reinforcement learning [116,117], to predict the motion of other robots; we may also use these technologies to generate initial values so as to speed up the convergence of SCP-based procedures; machine learning aided deadlock detection is also an important topic deserving further exploration. Third, with little $a\ priori$ work, another interesting topic is machine-learning-based attack detection and resilience. For example, based on proper machine learning models, we can predict whether the current motion of a robot or a system is correct.

Appendix A

List of Publications

1. **Yuan Zhou**, Hesuan Hu, Yang Liu, Shang-Wei Lin, Zuohua Ding. "A real-time and fully distributed approach to motion planning for multirobot systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Oct. 2017. http://ieeexplore.ieee.org/document/8055437/.

2. **Yuan Zhou**, Hesuan Hu, Yang Liu, and Zuohua Ding. "Collision and deadlock avoidance in multirobot systems: A distributed approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 7, pp. 1712–1726, Jul. 2017.

3. **Yuan Zhou**, Hesuan Hu, Yang Liu, Shang-Wei Lin, and Zuohua Ding. "A distributed approach to robust control of multi-robot systems," *Automatica*, vol. 98, pp. 1–13, Dec. 2018.

4. Zuohua Ding, **Yuan Zhou**, Mengchu Zhou. "Modeling self-adaptive software systems by fuzzy rules and Petri nets," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 2, pp. 967–984, Apr. 2018.

5. Jipeng Wang, Chunrong Pan, Hesuan Hu, Liang Li, and **Yuan Zhou**, "A cyclic scheduling approach to single-arm cluster tools with multiple wafer types and residency time constraints," *IEEE Transactions on Automation Science and Engineering*, Nov. 2018. https://ieeexplore.ieee.org/abstract/document/8543218.

6. Junyao Hou, Hesuan Hu, **Yuan Zhou**, Yang Liu. "Decentralized supervisory control of Generalized Mutual Exclusion Constraints in Petri Nets," *13th IEEE Conference on Automation Science and Engineering (CASE)*, Aug. 2017: 358-363.

7. Nan Du, Hesuan Hu, **Yuan Zhou**, Yang Liu. "Robust control of automated manufacturing systems with complex structures using Petri Nets," *13th IEEE Conference on Automation Science and Engineering (CASE)*, Aug. 2017: 364-369.

8. Xiaojun Wang, Hesuan Hu, **Yuan Zhou**, Yang Liu. "A robust control approach to automated manufacturing systems allowing failures and reworks with Petri nets," *13th IEEE Conference on Automation Science and Engineering (CASE)*, Aug. 2017: 370-375.

9. Jipeng Wang, Chunrong Pan, Hesuan Hu, **Yuan Zhou**. "Scheduling of single-arm cluster tools with multi-type wafers and shared PMs," *13th IEEE Conference on Automation Science and Engineering (CASE)*, Aug. 2017: 1046-1051.

Bibliography

[1] S. L. Smith, M. Schwager, and D. Rus, “Persistent robotic tasks: Monitoring and sweeping in changing environments,” *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 410–426, Apr. 2012.

[2] V. Kumar, D. Rus, and S. Singh, “Robot and sensor networks for first responders,” *IEEE Pervasive Computing*, vol. 3, no. 4, pp. 24–33, Oct. 2005.

[3] A. Marino, L. E. Parker, G. Antonelli, and F. Caccavale, “A decentralized architecture for multi-robot systems based on the null-space-behavioral control with application to multi-robot border patrolling,” *Journal of Intelligent & Robotic Systems*, vol. 71, no. 3-4, pp. 423–444, Sept. 2013.

[4] D. Portugal, G. Cabrita, B. D. Gouveia, D. C. Santos, and J. A. Prado, “An autonomous all terrain robotic system for field demining missions,” *Robotics and Autonomous Systems*, vol. 70, no. C, pp. 126–144, Aug. 2015.

[5] T. Breuer, G. R. G. Macedo, R. Hartanto *et al.*, “Johnny: An autonomous service robot for domestic environments,” *Journal of Intelligent & Robotic Systems*, vol. 66, no. 1-2, pp. 245–272, Apr. 2012.

[6] “World robotics 2018 edition,” https://ifr.org/free-downloads/, accessed: 2018-10-25.

[7] C. Kitts and M. Egerstedt, “Design, control, and applications of real-world multi-robot systems [from the guest editors],” *IEEE Robotics & Automation Magazine*, vol. 15, no. 1, p. 8, Mar. 2008.

[8] A. Khamis, A. Hussein, and A. Elmogy, “Multi-robot task allocation: A review of the state-of-the-art,” in *Cooperative Robots and Sensor Networks 2015*, A. Koubâa and J. R. Martínez-de Dios, Eds. Switzerland: Springer, 2015, pp. 31–51.

[9] J. H. Reif, “Complexity of the mover’s problem and generalizations,” in *20th Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, Oct. 1979, pp. 421–427.

[10] J. E. Hopcroft, J. T. Schwartz, and M. Sharir, “On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the“Warehouseman’s Problem”,” *The International Journal of Robotics Research*, vol. 3, no. 4, pp. 76–88, Dec. 1984.

[11] J. Reif and M. Sharir, “Motion planning in the presence of moving obstacles,” *Journal of the ACM*, vol. 41, no. 4, pp. 764–790, Jul. 1994.

[12] J.-C. Latombe, *Robot Motion Planning*. New York: Springer Science & Business Media, 1991.

[13] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. Boston, MA: MIT Press, 2005.

[14] S. M. LaValle, *Planning Algorithms*. Cambridge, UK: Cambridge University Press, 2006.

[15] M. Hoy, A. S. Matveev, and A. V. Savkin, “Algorithms for collision-free navigation of mobile robots in complex cluttered environments: A survey,” *Robotica*, vol. 33, no. 3, pp. 463–497, Mar. 2015.

[16] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka, “Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions,” *Transportation Research Part C: Emerging Technologies*, vol. 60, pp. 416–442, Nov. 2015.

[17] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, Mar. 2016.

[18] M. Kloetzer and C. Belta, "Temporal logic planning and control of robotic swarms by hierarchical abstractions," *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 320–330, Apr. 2007.

[19] A. M. Ayala, S. B. Andersson, and C. Belta, "Temporal logic motion planning in unknown environments," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, Nov. 2013, pp. 5279–5284.

[20] Y. Chen, J. Tůmová, A. Ulusoy, and C. Belta, "Temporal logic robot control based on automata learning of environmental dynamics," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 547–565, Apr. 2013.

[21] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013, pp. 5025–5032.

[22] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe LTL specifications," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Chicago, IL, Sept. 2014, pp. 1525–1532.

[23] J. Luo, H. Ni, and M. Zhou, "Control program design for automated guided vehicle systems via Petri nets," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 1, pp. 44–55, Jan. 2015.

[24] B. Hoxha and G. E. Fainekos, "Planning in dynamic environments through temporal logic monitoring," in *AAAI Workshop: Planning for Hybrid Systems*, vol. 16. Phoenix, Arizona: AAAI, Feb 2016, pp. 601–607.

[25] J. Tumova and D. V. Dimarogonas, "Multi-agent planning under local LTL specifications and event-based synchronization," *Automatica*, vol. 70, pp. 239–248, Aug 2016.

[26] Y. Zhou, H. Hu, Y. Liu, and Z. Ding, "Collision and deadlock avoidance in multirobot systems: A distributed approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 7, pp. 1712–1726, Jul. 2017.

[27] C. Mahulea and M. Kloetzer, "Robot planning based on boolean specifications using Petri net models," *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 2218–2225, 2018.

[28] M. Jiang, Z. Ding, M. Zhou, and Y. Zhou, "Formal modeling and verification of secure mobile agent systems," in *2015 IEEE international Conference on Automation Science and Engineering (CASE)*, Gothenburg, Sweden, Aug. 2015, pp. 545–550.

[29] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA: MIT Press, 2008.

[30] Y. Liu, J. Sun, and J. S. Dong, "Developing model checkers using PAT," in *8th International symposium on automated technology for verification and Analysis (ATVA 2010)*, Singapore, Sept. 2010, pp. 371–377.

[31] ——, "PAT 3: An extensible architecture for building multi-domain model checkers," in *2011 IEEE 22nd International Symposium on Software Reliability Engineering (ISSRE 2011)*, Hiroshima, Japan, Nov. 2011, pp. 190–199.

[32] C. Alexopoulos and P. M. Griffin, "Path planning for a mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 2, pp. 318–322, Mar. 1992.

[33] A. Kleiner, D. Sun, and D. Meyer-Delius, "Armo: Adaptive road map optimization for large robot teams," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, Sept. 2011, pp. 3276–3282.

[34] J. P. Van Den Berg and M. H. Overmars, "Roadmap-based motion planning in dynamic environments," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 885–897, Oct. 2005.

[35] D. Sun, A. Kleiner, and B. Nebel, "Behavior-based multi-robot collision avoidance," in *2014 IEEE International Conference on Robotics and Automation (I-CRA)*, Hong Kong, China, Jun. 2014, pp. 1668–1673.

[36] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, Oct. 1979.

[37] H. Rohnert, "Shortest paths in the plane with convex polygonal obstacles," *Information Processing Letters*, vol. 23, no. 2, pp. 71–76, Aug. 1986.

[38] H.-P. Huang and S.-Y. Chung, "Dynamic visibility graph for path planning," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS)*, vol. 3, Sendai, Japan, Sept. 2004, pp. 2813–2818.

[39] A. T. Rashid, A. A. Ali, M. Frasca, and L. Fortuna, "Path planning with obstacle avoidance based on visibility binary tree algorithm," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1440–1449, Dec. 2013.

[40] C. Ó'Dúnlaing and C. K. Yap, "A "retraction" method for planning the motion of a disc," *Journal of Algorithms*, vol. 6, no. 1, pp. 104–111, Mar. 1985.

[41] P. Bhattacharya and M. L. Gavrilova, "Roadmap-based path planning – Using the Voronoi diagram for a clearance-based shortest path," *IEEE Robotics & Automation Magazine*, vol. 15, no. 2, Jun. 2008.

[42] S. Kemna, J. G. Rogers, C. Nieto-Granda, S. Young, and G. S. Sukhatme, "Multi-robot coordination through dynamic Voronoi partitioning for informative adaptive sampling in communication-constrained environments," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, May 2017, pp. 2124–2130.

[43] S. Garrido, L. Moreno, M. Abderrahim, and F. Martin, “Path planning for mobile robot navigation using Voronoi diagram and fast marching,” in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Beijing, China, Oct. 2006, pp. 2376–2381.

[44] R. Geraerts and M. H. Overmars, “A comparative study of probabilistic roadmap planners,” in *Algorithmic Foundations of Robotics V*, J. D. Boissonnat, B. J, G. K, and H. S, Eds. Springer, 2004, pp. 43–57.

[45] P. Svestka, J. Latombe, and L. Overmars Kavraki, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[46] J. D. Marble and K. E. Bekris, “Asymptotically near-optimal planning with probabilistic roadmap spanners,” *IEEE Transactions on Robotics*, vol. 29, no. 2, pp. 432–444, Apr. 2013.

[47] T. Ort, L. Paull, and D. Rus, “Autonomous vehicle navigation in rural environments without detailed prior maps,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, May 2018, pp. 2040–2047.

[48] L. Zhang, Y. J. Kim, and D. Manocha, “A hybrid approach for complete motion planning,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, Oct. 2007, pp. 7–14.

[49] F. Lingelbach, “Path planning using probabilistic cell decomposition,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1. New Orleans, LA, USA, Apr. 2004, pp. 467–472.

[50] M. Kloetzer, C. Mahulea, and R. Gonzalez, “Optimizing cell decomposition path planning for mobile robots using different metrics,” in *2015 19th International Conference on System Theory, Control and Computing (ICSTCC)*, Cheile Gradistei, Romania, Oct. 2015, pp. 565–570.

[51] R. Gonzalez, M. Kloetzer, and C. Mahulea, "Comparative study of trajectories resulted from cell decomposition path planning approaches," in *2017 21st International Conference on System Theory, Control and Computing (ICSTCC)*, Sinaia, Romania, Oct 2017, pp. 49–54.

[52] B. Dugarjav, S.-G. Lee, T. B. Quang, K.-W. Gwak, and B. Lee, "Adaptive online cell decomposition with a laser range finder in unknown non-rectilinear environments," *International Journal of Precision Engineering and Manufacturing*, vol. 18, no. 4, pp. 487–495, Apr. 2017.

[53] Y. Guo and L. E. Parker, "A distributed and optimal motion planning approach for multiple mobile robots," in *2002 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3, Washington, DC, Aug. 2002, pp. 2612–2619.

[54] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, Mar. 2009.

[55] M. Pivtoraiko and A. Kelly, "Kinodynamic motion planning with state lattice motion primitives," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, Sept. 2011, pp. 2172–2179.

[56] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-UAV motion replanning for exploring unknown environments," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013, pp. 2452–2458.

[57] A. Krnjak, I. Draganjac, S. Bogdan, T. Petrović, D. Miklić, and Z. Kovačić, "Decentralized control of free ranging AGVs in warehouse environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May 2015, pp. 2034–2041.

[58] D. S. Yershov and S. M. LaValle, "Simplicial Dijkstra and A* algorithms for optimal feedback planning," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, USA, Sept. 2011, pp. 3862–3867.

[59] L. Janson, B. Ichter, and M. Pavone, "Deterministic sampling-based motion planning: Optimality, complexity, and performance," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 46–61, Jan. 2018.

[60] V. Pacelli, O. Arslan, and D. E. Koditschek, "Integration of local geometry and metric information in sampling-based motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, May 2018, pp. 3061–3068.

[61] Y. Ji, Y. Tanaka, Y. Tamura, M. Kimura, A. Umemura, Y. Kaneshima, H. Murakami, A. Yamashita, and H. Asama, "Adaptive motion planning based on vehicle characteristics and regulations for off-road UGVs," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 1, pp. 599–611, Jan. 2019.

[62] B. Burns and O. Brock, "Sampling-based motion planning with sensing uncertainty," in *2007 IEEE International Conference on Robotics and Automation (I-CRA)*, Roma, Italy, Apr. 2007, pp. 3313–3318.

[63] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, Feb. 2014.

[64] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: The driftless case," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May 2015, pp. 2368–2375.

[65] W. Sun, S. Patil, and R. Alterovitz, "High-frequency replanning under uncertainty using parallel sampling-based motion planning," *IEEE Transactions on Robotics*, vol. 31, no. 1, pp. 104–116, Feb. 2015.

[66] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *The International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, Dec. 1991.

[67] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Ames, IA, Tech. Rep. TR 98-11, Oct. 1998.

[68] O. Arslan, K. Berntorp, and P. Tsiotras, "Sampling-based algorithms for optimal motion planning using closed-loop prediction," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, May 2017, pp. 4991–4996.

[69] A. Yershova and S. M. LaValle, "Improving motion-planning algorithms by efficient nearest-neighbor searching," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 151–157, Feb. 2007.

[70] A. Bircher, K. Alexis, U. Schwesinger, S. Omari, M. Burri, and R. Siegwart, "An incremental sampling-based approach to inspection planning: the rapidly exploring random tree of trees," *Robotica*, vol. 35, no. 6, pp. 1327–1340, Jun. 2017.

[71] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[72] J. Ng and T. Bräunl, "Performance comparison of bug navigation algorithms," *Journal of Intelligent and Robotic Systems*, vol. 50, no. 1, pp. 73–84, Jan. 2007.

[73] V. Pavlov and A. Voronin, "The method of potential functions for coding constraints of the external space in an intelligent mobile robot," *Soviet Automatic Control*, vol. 17, no. 6, pp. 45–51, 1984.

[74] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, Oct. 1992.

[75] P. Vlantis, C. Vrohidis, C. P. Bechlioulis, and K. J. Kyriakopoulos, "Robot navigation in complex workspaces using harmonic maps," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, May 2018, pp. 1726–1731.

[76] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Autonomous Robots*, vol. 13, no. 3, pp. 207–222, Nov. 2002.

[77] S. A. Masoud and A. A. Masoud, "Constrained motion control using vector potential fields," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 30, no. 3, pp. 251–272, May 2000.

[78] G. Li, Y. Tamura, A. Yamashita, and H. Asama, "Effective improved artificial potential field-based regression search method for autonomous mobile robot path planning," *International Journal of Mechatronics and Automation*, vol. 3, no. 3, pp. 141–170, Jan. 2013.

[79] A. K. Pamosoaji and K.-S. Hong, "A path-planning algorithm using vector potential functions in triangular regions," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 4, pp. 832–842, Jul. 2013.

[80] H. G. Tanner and A. Boddu, "Multiagent navigation functions revisited," *IEEE Transactions on Robotics*, vol. 28, no. 6, pp. 1346–1359, Dec. 2012.

[81] D. V. Dimarogonas, S. G. Loizou, K. J. Kyriakopoulos, and M. M. Zavlanos, "A feedback stabilization and collision avoidance scheme for multiple independent non-point agents," *Automatica*, vol. 42, no. 2, pp. 229–243, Feb. 2006.

[82] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Stable flocking of mobile agents, Part I: Fixed topology," in *2003 IEEE 42nd International Conference on Decision and Control (CDC)*, vol. 2, Maui, HI, USA, Dec. 2003, pp. 2010–2015.

[83] J. Ji, A. Khajepour, W. W. Melek, and Y. Huang, "Path planning and tracking for vehicle collision avoidance based on model predictive control with multiconstraints," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 2, pp. 952–964, Feb. 2017.

[84] W.-B. Xu, X.-B. Chen, J. Zhao, and T.-Y. Huang, "A decentralized method using artificial moments for multi-robot path-planning," *International Journal of Advanced Robotic Systems*, vol. 10, no. 1, pp. 24:1–24:12, Jan. 2013.

[85] W.-B. Xu, X.-P. Liu, X. Chen, and J. Zhao, "Improved artificial moment method for decentralized local path planning of multirobots," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 6, pp. 2383–2390, Nov. 2015.

[86] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using the relative velocity paradigm," in *1993 IEEE International Conference on Robotics and Automation (ICRA)*, Atlanta, GA, USA, May 1993, pp. 560–565.

[87] ——, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, Jul. 1998.

[88] D. Wilkie, J. Van Den Berg, and D. Manocha, "Generalized velocity obstacles," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA, Oct. 2009, pp. 5573–5578.

[89] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, CA, May 2008, pp. 1928–1935.

[90] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha, "Smooth and collision-free navigation for multiple robots under differential-drive constraints," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, Oct. 2010, pp. 4584–4589.

[91] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Berlin, Heidelberg: Springer, 2011, pp. 3–19.

[92] J. Alonso-Mora, A. Breitenmoser, P. Beardsley, and R. Siegwart, "Reciprocal collision avoidance for multiple car-like robots," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, Minnesota, May 2012, pp. 360–366.

[93] J. Van Den Berg, J. Snape, S. J. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011, pp. 3475–3482.

[94] D. Bareiss and J. van den Berg, "Generalized reciprocal collision avoidance," *The International Journal of Robotics Research*, vol. 34, no. 12, pp. 1501–1514, Oct. 2015.

[95] J. Alonso-Mora, P. Beardsley, and R. Siegwart, "Cooperative collision avoidance for nonholonomic robots," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 404–420, Apr. 2018.

[96] P. Abichandani, G. Ford, H. Y. Benson, and M. Kam, "Mathematical programming for multi-vehicle motion planning problems," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, Saint Paul, MN, May 2012, pp. 3315–3322.

[97] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Algarve, Portugal, Oct. 2012, pp. 1917–1922.

[98] Y. Chen, M. Cutler, and J. P. How, "Decoupled multiagent path planning via incremental sequential convex programming," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May 2015, pp. 5954–5961.

[99] R. Deits and R. Tedrake, "Efficient mixed-integer planning for UAVs in cluttered environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May 2015, pp. 42–49.

[100] H. Fukushima, K. Kon, and F. Matsuno, "Model predictive formation control using branch-and-bound compatible with collision avoidance problems," *IEEE Transactions on Robotics*, vol. 29, no. 5, pp. 1308–1317, May 2013.

[101] S. K. Gan, R. Fitch, and S. Sukkarieh, "Real-time decentralized search with inter-agent collision avoidance," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, Saint Paul, MN, Jun. 2012, pp. 504–510.

[102] J. Park, S. Karumanchi, and K. Iagnemma, "Homotopy-based divide-and-conquer strategy for optimal trajectory planning via mixed-integer programming," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1101–1115, Aug. 2015.

[103] D. Morgan, S.-J. Chung, and F. Y. Hadaegh, "Model predictive control of swarms of spacecraft using sequential convex programming," *Journal of Guidance, Control, and Dynamics*, vol. 37, no. 6, pp. 1725–1740, Nov. 2014.

[104] Z. Li, J. Deng, R. Lu, Y. Xu, J. Bai, and C.-Y. Su, "Trajectory-tracking control of mobile robot systems incorporating neural-dynamic optimized model predictive approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 6, pp. 740–749, Jun. 2016.

[105] Y. Zhou, H. Hu, Y. Liu, S.-W. Lin, and Z. Ding, "A real-time and fully distributed approach to motion planning for multirobot systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Oct. 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8055437

[106] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *2014 IEEE 53rd Annual Conference on Decision and Control (CDC)*, Los Angeles, CA, USA, Dec. 2014, pp. 81–87.

[107] M. Egerstedt, "Motion planning and control of mobile robots," Ph.D. dissertation, Department of Mathematics, KTH, 2000.

[108] J.-W. Choi, R. E. Curry, and G. H. Elkaim, "Continuous curvature path generation based on Bézier curves for autonomous vehicles." *International Journal of Applied Mathematics*, vol. 40, no. 2, 2010.

[109] F. Gao, W. Wu, Y. Lin, and S. Shen, "Online safe trajectory generation for quadrotors using fast marching method and Bernstein basis polynomial," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, 2018, pp. 344–351.

[110] F. Gómez-Bravo, F. Cuesta, A. Ollero, and A. Viguria, "Continuous curvature path generation based on β-spline curves for parking manoeuvres," *Robotics and Autonomous Systems*, vol. 56, no. 4, pp. 360–372, Apr. 2008.

[111] J. J. Liang, H. Song, B. Y. Qu, and Z. F. Liu, "Comparison of three different curves used in path planning problems based on particle swarm optimizer," *Mathematical Problems in Engineering*, vol. 2014, Apr. 2014, Article ID 623156, 15 pages.

[112] A. Konar, I. Goswami, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved Q-learning for path planning of a mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 5, pp. 1141–1153, Sept. 2013.

[113] P. Rakshit, A. Konar, P. Bhowmik, I. Goswami, S. Das, L. C. Jain, and A. K. Nagar, "Realization of an adaptive memetic algorithm using differential evolution and Q-learning: A case study in multirobot path planning," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 4, pp. 814–831, Jul. 2013.

[114] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, "Socially compliant mobile robot navigation via inverse reinforcement learning," *The International Journal of Robotics Research*, vol. 35, no. 11, pp. 1289–1307, Jul. 2016.

[115] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, Canada, Sept. 2017, pp. 1343–1350.

[116] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (I-CRA)*, Singapore, May 2017, pp. 285–292.

[117] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, Oct. 2018, pp. 3052–3059.

[118] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, May 2018, pp. 2034–2039.

[119] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 4, no. 1, pp. 1–103, 2010.

[120] J. A. DeCastro, J. Alonso-Mora, V. Raman, D. Rus, and H. Kress-Gazit, "Collision-free reactive mission and motion planning for multi-robot systems," in *Robotics Research*, A. Bicchi and W. Burgard, Eds. Springer, 2018, pp. 459–476.

[121] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems," *The International Journal of Robotics Research*, vol. 37, no. 7, pp. 818–838, Jun 2018.

[122] M. Guo, J. Tumova, and D. V. Dimarogonas, "Hybrid control of multi-agent systems under local temporal tasks and relative-distance constraints," in *54th IEEE Conference on Decision and Control (CDC)*, Osaka, Japan, Dec. 2015, pp. 1701–1706.

[123] M. Guo, C. P. Bechlioulis, K. J. Kyriakopoulos, and D. V. Dimarogonas, "Hybrid control of multiagent systems with contingent temporal tasks and prescribed formation constraints," *IEEE Transactions on Control of Network Systems*, vol. 4, no. 4, pp. 781–792, Dec. 2017.

[124] N. Malone, H.-T. Chiang, K. Lesser, M. Oishi, and L. Tapia, "Hybrid dynamic moving obstacle avoidance using a stochastic reachable set-based potential field," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1124–1138, Oct. 2017.

[125] M. Egerstedt and X. Hu, "A hybrid control approach to action coordination for mobile robots," *Automatica*, vol. 38, no. 1, pp. 125–130, Jan. 2002.

[126] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *Proceedings of the 44th IEEE Conference on Decision and Control (CDC)*, Seville, Spain, Dec. 2005, pp. 4885–4890.

[127] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local LTL specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, Feb. 2015.

[128] S. Akella and S. Hutchinson, "Coordinating the motions of multiple robots with specified trajectories," in *2002 IEEE International Conference on Robotics and Automation (ICRA)*, Washington, DC, May 2002, pp. 624–631.

[129] K.-D. Kim and P. R. Kumar, "An MPC-based approach to provable system-wide safety and liveness of autonomous ground traffic," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3341–3356, Dec. 2014.

[130] D. E. Soltero, S. L. Smith, and D. Rus, "Collision avoidance for persistent monitoring in multi-robot systems with intersecting trajectories," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, Sept. 2011, pp. 3645–3652.

[131] E. J. Rodríguez-Seda, C. Tang, M. W. Spong, and D. M. Stipanović, "Trajectory tracking with collision avoidance for nonholonomic vehicles with acceleration constraints and limited sensing," *The International Journal of Robotics Research*, vol. 33, no. 12, pp. 1569–1592, Oct. 2014.

[132] X. Wang, M. Kloetzer, C. Mahulea, and M. Silva, "Collision avoidance of mobile robots by using initial time delays," in *2015 IEEE 54th Annual Conference on Decision and Control (CDC)*, Osaka, Japan, Dec. 2015, pp. 324–329.

[133] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-92-01, Jan. 1992.

[134] H. Andersen, Z. J. Chong, Y. H. Eng, S. Pendleton, and M. H. Ang, “Geometric path tracking algorithm for autonomous driving in pedestrian environment,” in *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, Banff, AB, Canada, Jul. 2016, pp. 1669–1674.

[135] T. Yamasaki, H. Takano, and Y. Baba, “Robust path-following for UAV using pure pursuit guidance,” in *Aerial Vehicles*, T. M. Lam, Ed. InTech, Jan. 2009, pp. 671–690.

[136] A. Al-Mayyahi, W. Wang, and P. Birch, “Path tracking of autonomous ground vehicle based on fractional order PID controller optimized by PSO,” in *2015 IEEE 13th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, Herl’any, Slovakia, Jan. 2015, pp. 109–114.

[137] B. Hu, G. K. Mann, and R. G. Gosine, “New methodology for analytical and optimal design of fuzzy PID controllers,” *IEEE Transactions on Fuzzy Systems*, vol. 7, no. 5, pp. 521–539, Oct. 1999.

[138] K. Shojaei, “Neural adaptive PID formation control of car-like mobile robots without velocity measurements,” *Advanced Robotics*, vol. 31, no. 18, pp. 947–964, Sept. 2017.

[139] G. Antonelli, S. Chiaverini, and G. Fusco, “A fuzzy-logic-based approach for mobile robot path tracking,” *IEEE Transactions on Fuzzy Systems*, vol. 15, no. 2, pp. 211–221, Apr. 2007.

[140] E. Maalouf, M. Saad, and H. Saliah, “A higher level path tracking controller for a four-wheel differentially steered mobile robot,” *Robotics and Autonomous Systems*, vol. 54, no. 1, pp. 23–33, Jan. 2006.

[141] E. Kim, J. Kim, and M. Sunwoo, “Model predictive control strategy for smooth path tracking of autonomous vehicles with steering actuator dynamics,” *International Journal of Automotive Technology*, vol. 15, no. 7, pp. 1155–1164, Dec. 2014.

[142] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, Oct. 2009.

[143] P. F. Lima, M. Nilsson, M. Trincavelli, J. Mårtensson, and B. Wahlberg, "Spatial model predictive control for smooth and accurate steering of an autonomous truck," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 4, pp. 238–250, Dec. 2017.

[144] H. Ashrafiuon, S. Nersesov, and G. Clayton, "Trajectory tracking control of planar underactuated vehicles," *IEEE Transactions on Automatic Control*, vol. 62, no. 4, pp. 1959–1965, Apr. 2017.

[145] E. G. Coffman, M. Elphick, and A. Shoshani, "System deadlocks," *ACM Computing Surveys*, vol. 3, no. 2, pp. 67–78, Jun. 1971.

[146] Z. Li, N. Wu, and M. Zhou, "Deadlock control of automated manufacturing systems based on Petri nets−A literature review," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 4, pp. 437–462, Jul. 2012.

[147] M. Uzam and M. C. Zhou, "An iterative synthesis approach to Petri net-based deadlock prevention policy for flexible manufacturing systems," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 37, no. 3, pp. 362–371, May 2007.

[148] Z. W. Li and M. C. Zhou, "Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 34, no. 1, pp. 38–51, Jan. 2004.

[149] H. Chen, N. Q. Wu, and M. C. Zhou, "A novel method for deadlock prevention of AMS by using resource-oriented Petri nets," *Information Sciences*, vol. 363, pp. 178–189, Oct. 2016.

[150] K. Xing, F. Wang, M. C. Zhou, H. Lei, and J. Luo, "Deadlock characterization and control of flexible assembly systems with Petri nets," *Automatica*, vol. 87, pp. 358–364, Jan. 2018.

[151] J. Ye, M. Zhou, Z. Li, and A. Al-Ahmari, "Structural decomposition and decentralized control of Petri nets," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 8, pp. 1360–1369, Aug. 2018.

[152] J. Luo, Z. Liu, M. Zhou, and K. Xing, "Deadlock-free scheduling of flexible assembly systems based on Petri nets and local search," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Sept. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8457481

[153] H. R. Golmakani, J. K. Mills, and B. Benhabib, "Deadlock-free scheduling and control of flexible manufacturing cells using automata theory," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 36, no. 2, pp. 327–337, Mar. 2006.

[154] A. Ramirez-Serrano and B. Benhabib, "Supervisory control of multi-workcell manufacturing systems with shared resources," in *2000 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3, San Francisco, CA, USA, Apr. 2000, pp. 2847–2852.

[155] H. Hu, Y. Liu, and M. Zhou, "Maximally permissive distributed control of large scale automated manufacturing systems modeled with petri nets," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 5, pp. 2026–2034, Feb. 2015.

[156] H. Cho, T. Kumaran, and R. A. Wysk, "Graph-theoretic deadlock detection and resolution for flexible manufacturing systems," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 413–421, Jun. 1995.

[157] M. Jager and B. Nebel, "Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots," in *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, Maui, HI, Oct. 2001, pp. 1213–1219.

[158] R. L. Moorthy, W. Hock-Guan, N. Wing-Cheong, and T. Chung-Piaw, "Cyclic deadlock prediction and avoidance for zone-controlled AGV system," *International Journal of Production Economics*, vol. 83, no. 3, pp. 309–324, Mar. 2003.

[159] H. Hu and Y. Liu, "Supervisor synthesis and performance improvement for automated manufacturing systems by using Petri nets," *IEEE Transactions on Industrial Informatics*, vol. 2, no. 11, pp. 450–458, Apr. 2015.

[160] H. Hu and M. Zhou, "A Petri net-based discrete-event control of automated manufacturing systems with assembly operations," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 2, pp. 513–524, Mar. 2015.

[161] C.-C. Lee and J. T. Lin, "Deadlock prediction and avoidance based on Petri nets for zone-control automated guided vehicle systems," *International Journal of Production Research*, vol. 33, no. 12, pp. 3249–3265, Dec. 1995.

[162] S. A. Reveliotis and E. Roszkowska, "Conflict resolution in free-ranging multivehicle systems: A resource allocation paradigm," *IEEE Transactions on Robotics*, vol. 27, no. 2, pp. 283–296, Apr. 2011.

[163] L. Kalinovcic, T. Petrovic, S. Bogdan, and V. Bobanac, "Modified banker's algorithm for scheduling in multi-agv systems," in *IEEE Conference on Automation Science and Engineering*, Trieste, Italy, Oct. 2011, pp. 351–356.

[164] M. P. Fanti, A. M. Mangini, G. Pedroncelli, and W. Ukovich, "Decentralized deadlock-free control for AGV systems," in *2015 American Control Conference (ACC)*, Chicago, IL, USA, Jul. 2015, pp. 2414–2419.

[165] A. Yalcin and T. O. Boucher, "Deadlock avoidance in flexible manufacturing systems using finite automata," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 4, pp. 424–429, Aug. 2000.

[166] Y. Yang, H. Hu, and Y. Liu, "A Petri net-based distributed control of automated manufacturing systems with assembly operations," in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. Gothenburg, Sweden, Aug. 2015, pp. 1090–1097.

[167] J. Hou, H. Hu, Y. Zhou, and Y. Liu, “Decentralized supervisory control of generalized mutual exclusion constraints in Petri nets,” in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, Xi’an, China, Aug. 2017, pp. 358–363.

[168] Y. Yang, H. Hu, and Y. Liu, “A distributed approach to automated manufacturing systems with complex structures using Petri nets,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, May 2017, pp. 3016–3023.

[169] H. Hu, R. Su, M. Zhou, and Y. Liu, “Polynomially complex synthesis of distributed supervisors for large-scale AMSs using Petri nets,” *IEEE Transactions on Control Systems Technology*, vol. 24, no. 5, pp. 1610–1622, Dec. 2016.

[170] H. Hu, Y. Yang, Y. Liu, and N. Du, “Critical stages and their application in large scale automated manufacturing systems via Petri nets,” in *2016 European Control Conference (ECC)*, Jun. 2016, pp. 2337–2344.

[171] S. Reveliotis, *Real-time Management of Resource Allocation Systems: A Discrete Event Systems Approach*. New York: Springer Science & Business Media, 2006.

[172] E. W. Dijkstra, “Cooperating sequential processes,” in *The Origin of Concurrent Programming*, F. Genuys, Ed. New York: Academic, 1968, pp. 65–138.

[173] M. B. Dias, M. Zinck, R. Zlot, and A. Stentz, “Robust multirobot coordination in dynamic environments,” in *2004 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, New Orleans, LA, Apr. 2004, pp. 3435–3442.

[174] K. Goldberg and B. Chen, “Collaborative control of robot motion: Robustness to error,” in *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, Maui, HI, Oct. 2001, pp. 655–660.

[175] N. Hazon and G. A. Kaminka, “On redundancy, efficiency, and robustness in coverage for multiple robots,” *Robotics and Autonomous Systems*, vol. 56, no. 12, pp. 1102–1114, Dec. 2008.

[176] L. E. Parker *et al.*, "ALLIANCE: An architecture for fault tolerant multirobot cooperation," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 220–240, Apr. 1998.

[177] W. Wu and F. Zhang, "Robust cooperative exploration with a switching strategy," *IEEE Transactions on Robotics*, vol. 28, no. 4, pp. 828–839, Aug. 2012.

[178] M. Dogar, R. A. Knepper, A. Spielberg, C. Choi, H. I. Christensen, and D. Rus, "Multi-scale assembly with robot teams," *The International Journal of Robotics Research*, vol. 34, no. 13, pp. 1645–1659, Nov. 2015.

[179] M. Hofbaur, J. Köb, G. Steinbauer, and F. Wotawa, "Improving robustness of mobile robots using model-based reasoning," *Journal of Intelligent and Robotic Systems*, vol. 48, no. 1, pp. 37–54, Jan. 2007.

[180] T. Preisler and W. Renz, "Scalability and robustness analysis of a multi-agent based self-healing resource-flow system," in *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Wroclaw, Poland, Sept. 2012, pp. 1216–1268.

[181] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, Jul. 2013.

[182] Z. Liu, H. Wang, L. Xu, Y.-H. Liu, J. Lu, and W. Chen, "A failure-tolerant approach to synchronous formation control of mobile robots under communication delays," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, May 2018, pp. 1661–1666.

[183] L. Blackmore, H. Li, and B. Williams, "A probabilistic approach to optimal robust path planning with obstacles," in *2006 American Control Conference (ACC)*, Minneapolis, MN, Jun. 2006, pp. 2831–2837.

[184] L. Blackmore, M. Ono, and B. C. Williams, "Chance-constrained optimal path planning with obstacles," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1080–1094, Dec. 2011.

[185] Z. Li, J. Li, and Y. Kang, "Adaptive robust coordinated control of multiple mobile manipulators interacting with rigid environments," *Automatica*, vol. 46, no. 12, pp. 2028–2034, Dec. 2010.

[186] C.-S. Chiu, K.-Y. Lian, and T.-C. Wu, "Robust adaptive motion/force tracking control design for uncertain constrained robot manipulators," *Automatica*, vol. 40, no. 12, pp. 2111–2119, Dec. 2004.

[187] S. Liemhetcharat and M. Veloso, "Forming an effective multi-robot team robust to failures," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, Nov. 2013, pp. 5240–5245.

[188] F. Mastrogiovanni, A. Sgorbissa, and R. Zaccaria, "Robust navigation in an unknown environment with minimal sensing and representation," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 1, pp. 212–229, Jan. 2009.

[189] Z. Sun, L. Dai, K. Liu, Y. Xia, and K. H. Johansson, "Robust MPC for tracking constrained unicycle robots with additive disturbances," *Automatica*, vol. 90, pp. 172–184, Apr. 2018.

[190] X. Wang, H. Hu, Y. Zhou, and Y. Liu, "A robust control approach to automated manufacturing systems allowing failures and reworks with Petri nets," in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, Xi'an, China, Aug. 2017, pp. 370–375.

[191] N. Du, H. Hu, and Y. Liu, "Robust control of automated manufacturing systems with assembly operations using petri nets," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, May 2016, pp. 3632–3638.

[192] N. Du, H. Hu, Y. Zhou, and Y. Liu, "Robust control of automated manufacturing systems with complex structures using Petri nets," in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, Xi'an, China, Aug. 2017, pp. 364–369.

[193] E. F. Camacho and C. B. Alba, *Model Predictive Control*, 2nd ed. London, UK: Springer Science & Business Media, 2007.

[194] M. Bellare and P. Rogaway, "The complexity of approximating a nonlinear program," in *Complexity in Numerical Optimization*. World Scientific, 1993, pp. 16–32.

[195] K. Svanberg, "The method of moving asymptotes – A new method for structural optimization," *International Journal for Numerical Methods in Engineering*, vol. 24, no. 2, pp. 359–373, Feb. 1987.

[196] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004.

[197] J. Duchi, "Sequential convex programming," 2018. [Online]. Available: http://web.stanford.edu/class/ee364b/lectures/seq_notes.pdf

[198] J. Van Den Berg and M. Overmars, "Planning time-minimal safe paths amidst unpredictably moving obstacles," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1274–1294, Nov. 2008.

[199] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," Mar. 2014. [Online]. Available: http://cvxr.com/cvx

[200] H. Kress-Gazit, M. Lahijanian, and V. Raman, "Synthesis for robots: Guarantees and feedback for robot behavior," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 211–236, May 2018.

[201] R. N. Smith, M. Schwager, S. L. Smith, B. H. Jones, D. Rus, and G. S. Sukhatme, "Persistent ocean monitoring with underwater gliders: Adapting sampling resolution," *Journal of Field Robotics*, vol. 28, no. 5, pp. 714–741, Aug. 2011.

[202] K. Kim, G. Fainekos, and S. Sankaranarayanan, "On the minimal revision problem of specification automata," *The International Journal of Robotics Research*, vol. 34, no. 12, pp. 1515–1535, Aug. 2015.

[203] E. Roszkowska and S. Reveliotis, “A distributed protocol for motion coordination in free-range vehicular systems,” *Automatica*, vol. 49, no. 6, pp. 1639–1653, Jun. 2013.

[204] J. L. Gross, J. Yellen, and P. Zhang, *Handbook of Graph Theory*, 2nd ed. Boca Raton, Florida, USA: CRC Press, 2013.

[205] Q. T. Dinh and M. Diehl, “Local convergence of sequential convex programming for nonconvex optimization,” in *Recent Advances in Optimization and its Applications in Engineering*, M. Diehl, F. Glineur, E. Jarlebring, and M. Michiels, Eds., 2010, pp. 93–102.

[206] S. M. LaValle and S. A. Hutchinson, “Optimal motion planning for multiple robots having independent goals,” *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 912–925, Dec. 1998.

[207] S. M. LaValle, “Robot motion planning: A game-theoretic foundation,” *Algorithmica*, vol. 26, no. 3-4, pp. 430–465, Apr. 2000.

[208] Z. Ding, T. Xu, T. Ye, and Y. Zhou, “Online prediction and improvement of reliability for service oriented systems,” *IEEE Transactions on Reliability*, vol. 65, no. 3, pp. 1133–1148, Sept. 2016.

[209] Z. Ding, Y. Zhou, G. Pu, and M. Zhou, “Online failure prediction for railway transportation systems based on fuzzy rules and data analysis,” *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 1143–1158, Sept. 2018.

[210] D. Niyato and E. Hossain, “A game theoretic analysis of service competition and pricing in heterogeneous wireless access networks,” *IEEE Transactions on Wireless Communications*, vol. 7, no. 12, pp. 5150–5155, Dec. 2008.

[211] D. Niyato, P. Wang, E. Hossain, W. Saad, and A. Hjorungnes, “Exploiting mobility diversity in sharing wireless access: A game theoretic approach,” *IEEE Transactions on Wireless Communications*, vol. 9, no. 12, pp. 3866–3877, Dec. 2010.

[212] K. Zhu, E. Hossain, and D. Niyato, “Pricing, spectrum sharing, and service selection in two-tier small cell networks: A hierarchical dynamic game approach,” *IEEE Transactions on Mobile Computing*, vol. 13, no. 8, pp. 1843–1856, Dec. 2014.

[213] Z. Ding, Y. Zhou, and M. Zhou, “Modeling self-adaptive software systems with learning Petri nets,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 4, pp. 483–498, Apr. 2016.

[214] ——, “Modeling self-adaptive software systems by fuzzy rules and Petri nets,” *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 2, pp. 967–984, Apr. 2018.