Collision and Deadlock Avoidance in Multirobot Systems: A Distributed Approach

Zhou, Yuan; Hu, Hesuan; Liu, Yang; Ding, Zuohua

2017

Zhou, Y., Hu, H., Liu, Y., & Ding, Z. (2017). Collision and Deadlock Avoidance in Multirobot Systems: A Distributed Approach. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 47(7), 1712-1726.

https://hdl.handle.net/10356/83439

https://doi.org/10.1109/TSMC.2017.2670643

Collision and Deadlock Avoidance in Multirobot Systems: A Distributed Approach

Yuan Zhou, Hesuan Hu, *Senior Member, IEEE*, Yang Liu, and Zuohua Ding, *Member, IEEE*

***Abstract*—Collision avoidance is a critical problem in motion planning and control of multirobot systems. Moreover, it may induce deadlocks during the procedure to avoid collisions. In this paper, we study the motion control of multirobot systems where each robot has its own predetermined and closed path to execute persistent motion. We propose a real-time and distributed algorithm for both collision and deadlock avoidance by repeatedly stopping and resuming robots. The motion of each robot is first modeled as a labeled transition system, and then controlled by a distributed algorithm to avoid collisions and deadlocks. Each robot can execute the algorithm autonomously and real-timely 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. A set of simulations for a system with four robots are carried out in MATLAB. The results also validate the effectiveness of our algorithm.**

***Index Terms*—Collision and deadlock avoidance, discrete event systems, distributed algorithm, maximally permissive, motion control, multirobot systems.**

I. Introduction

COMPARED with their single-robot counterparts, multirobot systems become increasingly prevailing thanks to their benefits like wide coverage, diverse functionality, and strong flexibility [18]. Besides, with the collective behavior of multiple robots, a multirobot system has the enthralling capability to accomplish sophisticated tasks [20]. Multirobot systems have been applied in many areas [7], [20], [26], [34], such as surveillance tasks, reconnaissance missions, security service, and so on.

However, a common but essential and challenging problem for the motion planning of multirobot systems is to avoid collisions, including collisions with obstacles and/or other robots. Many heuristic methods have been proposed to address this problem, such as optimization programming [8], [9], [24], reciprocal collision avoidance [1], potential fields [27], sampling-based methods [3], formal methods based on linear temporal logic (LTL)/CTL [16], [33], and so on. Generally, there are two basic ideas that are applied. The first one focuses on the systems where robots have flexible paths and can change their paths at any time. It usually avoids collisions by planning/replanning collision-free paths so that different robots can be at different places at the same time. This idea concentrates on the change of robots's trajectories. The second one is for the systems where robots have fixed paths, which are limited by the environmental infrastructure, e.g., the highways in a city are predetermined, or are generated by the off-line planners using aforementioned methods. Robots cannot change their paths. Thus, proper motion controllers are designed, e.g., assigning robots with different initial delays, so that each robot can traverse a same location at a different time. This idea focuses on the time to traverse a same position. Note that in some multirobot systems, the two ideas can be combined for motion planning.

A nice way to perform a multirobot system is that each robot can change its trajectory freely. However, because of the limitation of the environment and infrastructure, sometimes the paths of robots are not allowed to change. Such scenarios are common in the transport systems and warehouses. For example, autonomous cars are required to move along particular circular roads to monitor the real-time traffic condition in a city; unmanned aerial vehicles are used to fly on determined authorized airways to monitor the air quality, such as temperature, PM2.5, and haze. For such scenarios, in practice, we always need to make sure that there are no static obstacles on the paths.

In this paper, we focus on the multirobot systems where each robot needs to move along a predetermined, fixed, and closed path. The paths are assumed to be static obstacle-free. Robots in the same system are homogeneous and are required to do persistent motion. Such systems are first studied in [34] and [35]. Smith *et al.* [34] investigated the design of the speed controllers of robots to perform persistent tasks without considering collisions or deadlocks. In [35], they further consider deadlocks since the paths intersect with each

Manuscript received June 9, 2016; revised August 4, 2016; accepted February 2, 2017. This work was supported in part by the Natural Science Foundation of China under Grant 61573265, Grant 61203037, Grant 51305321, Grant 61210004, and Grant 61170015, in part by the Fundamental Research Funds for the Central Universities under Grant K7215581201, Grant K5051304004, and Grant K5051304021, in part by the New Century Excellent Talents in University under Grant NCET-12-0921, in part by the Academic Research Fund Tier 1 by Ministry of Education in Singapore under Grant 2014-T1-001-147, and in part by the Academic Research Fund Tier 2 by Ministry of Education in Singapore under Grant MOE2015-T2-2-049. This paper was recommended by Associate Editor C. J. F. Silvestre.

Y. Zhou and Y. Liu are with the School of Computer Science and Engineering, College of Engineering, Nanyang Technological University, Singapore 639798 (e-mail: y.zhou@ntu.edu.sg; yangliu@ntu.edu.sg).

H. Hu is with the School of Computer Science and Engineering, College of Engineering, Nanyang Technological University, Singapore 639798, and also with the School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China (e-mail: huhesuan@gmail.com).

Z. Ding is with the School of Information Sciences, Zhejiang Sci-Tech University, Hangzhou 310018, China (e-mail: zouhuading@hotmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2017.2670643

other. They divide all collision locations into several disjoint collision zones. This means for any robot, there exists at least one safe location between any two collision zones. Thus, collisions and physical deadlocks can be avoided at the same time by repeatedly stopping and resuming robots such that at most one robot can be in an arbitrary collision zone. However, this is too conservative and causes low performance of the system. In order to improve the performance, some stopping policies are proposed. With these policies, each robot independently makes the decision to move or to wait for another one. Thus, decision deadlocks can occur because the pair-wise decisions made by some robots may contradict with each other. However, physically the robots can still move forward. Hence, even if it occurs, a decision deadlock can be resolved easily by resuming one of the robots.

This paper is an alternative improvement of that in [35]. The main difference is the way to deal with collision regions. We alternatively consider the collision segments directly. It can reduce conservation. However, there may exist two or more adjacent collision regions in which a robot collides with different robots. Thus, it may cause physical deadlocks during collision avoidance. This means that some robots, if not all, cannot move any more physically. Such deadlocks are more complicated and dangerous. They should be detected and resolved early because once a physical deadlock occurs, the system has to be redesigned and started over.

Researchers have proposed several methods to avoid collisions and deadlocks, such as Petri nets, automata, graph theory, and time-delay methods. However, most of the existing work considers the situation that robots move from the initial positions to the target positions, rather than persistent motion. We consider robots doing persistent motion. This means robots should repeatedly traverse their paths. Thus, the existing methods cannot be used directly. For example, Petri net-based method can cause state explosion since each time a robot needs to check the whole state space to determine whether it is safe to move back to its current state; we may not find proper time delays for robots since the motion time is infinite. Moreover, these methods are with poor scalability.

In this paper, we propose a distributed algorithm to avoid deadlocks by repeatedly stopping and resuming robots. Our approach relates to control of discrete-event systems (DESs). We first model the robot motion by labeled transition systems (LTSs) based on the intersections of their paths. Then a distributed algorithm is proposed to avoid collisions real-timely among different robots. Under this algorithm, each robot can execute its own mechanism autonomously to 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. Hence, an improved distributed algorithm is proposed to avoid not only collisions but also deadlocks. In the improved algorithm, a procedure is added to check whether the one-step move of a robot can cause a deadlock. If "yes," the algorithm will control the robot to stop its motion. A set of simulations are carried out in MATLAB. The results validate the effectiveness of the algorithm.

The main contribution of this paper is a real-time and distributed algorithm to avoid collisions and physical deadlocks in multirobot 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 the number of robots in the system. Thus, it is available to add 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 multirobot 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.

The remaining part of this paper is organized as follows. In Section II, we briefly describe some existing related work. In Section III, we give the LTS models for the motion of a single robot and the entire system. The persistent motion problem of the system is also stated in this section. A distributed algorithm for collision avoidance is presented in Section IV. In Section V, we propose an improved distributed algorithm for both collision and deadlock avoidance. The simulation results and implementation are described in Section VI. Section VII gives some discussion about this paper. Finally, the conclusion and some future work are discussed in Section VIII.

II. Related Work

Motion planning for multiple robots has been given a great attention both in academia and in industry. The main objective is to command each robot finish its required tasks without causing any collisions with external obstacles and/or other robots. Despite its appearance to be simple, this problem can be challenging to solve appropriately. Hopcroft *et al.* [13] show that even a simplified 2-D case of this problem is PSPACE-hard. Many researchers have made great effort to the solution of collision and deadlock avoidance in multirobot systems and carried out much fruitful work, such as [1]–[3], [5], [9], [10], [12], [14]–[17], [21]–[23], [25], [27], [28], [32], [33], [35], and [37], and the references therein.

Generally, researchers focus on the motion planning in two different scenarios of multirobot systems: 1) robots can change their paths and 2) robots are fixed on prescribed paths.

For the first scenario, by planning/replanning the motion paths of robots, each robot can deviate from its prescribed path so as to circumvent obstacles and other robots [1], [3], [8], [9], [12], [14]–[17], [21], [27], [32], [33].

Gan *et al.* [9] used a decentralized gradient-based optimization approach to avoiding interagent collisions in a team of mobile autonomous agents. The safety distance constraints are dynamically enforced in the optimization process of the agents' real-time group mission. Thus, solving the distributed optimization problem of each robot can generate a real-time internal collision-free path.

Kloetzer and Belta [16] proposed a hierarchical framework for planning and control of arbitrarily large groups of robots with polyhedral velocity bounds moving in polygonal environments with polygonal obstacles. In their approach, the inter-robot collision avoidance is described by LTL specifications. Thus, under the framework, only the paths that satisfy the LTL specifications can be generated, thereby guaranteeing no collisions.

However, the methods based on this idea can only be applied by the systems where robots can change their trajectories at any time. Thus, they cannot be applied in this paper since all the robots in our system have fixed prescribed paths.

Usually, the idea to avoid collisions in the second scenario is that to avoid collisions is to make robots traverse the same location at different times [2], [34]–[36]. Thus, collisions among different robots are checked and avoided by controlling the robots to traverse the same location at different times. The challenge is to optimize the performance of the system such that robots can move as smoothly as possible. For example, Soltero *et al.* [35] avoided collisions by stopping and resuming robots repeatedly. Wang *et al.* [36] also assigned robots different optimal initial time delays using the mixed integer linear programming optimization so that each robot can move from the initial position to the goal position without causing collisions.

There is some work combining these two ideas to control robot motion in the first kind of systems. It usually contains two phases. First, an external obstacle-free path for each robot is generated. Second, collisions among different robots are checked and avoided by controlling the robots to traverse the same location at different times. For example, in [10], the D^* search algorithm is first applied to produce an obstacle-free path independently for each robot. Once they are obtained, the paths are fixed. Then, each robot is associated with an optimal time delay as required to avoid collisions with other robots. Note that the premise of such methods is that the system can plan paths freely.

III. Multirobot Systems and Problem Statement

In this section, we focus on the formal definition of the persistent motion problem of a multirobot system. First, we give the description of the multirobot systems. Second, we use LTSs to model the motion of the system for further analysis. Third, we give the formalized problem statement of the persistent motion control of such systems. The following notations are used. N is the number of robots in the system, $\mathbb{N}_N = \{1, 2, \ldots, N\}$, and $r_i, i \in \mathbb{N}_N$, is the ith robot.

A. Description of the Multirobot Systems With Fixed Paths

In this section, we give a brief description of the multirobot systems where each robot has a fixed path.

Definition 1 (Path): The path of robot r_i, denoted as $\mathscr{P}^i$, is a simple, closed, and directed curve defined by the parameter equation $\mathscr{P}^i = P(\theta), \theta \in [0, 1]$ and $P(0) = P(1)$. The robot's motion direction is given by increasing θ.

Remark 1: For an automated ground vehicle, $\mathscr{P}^i$ is a curve in the 2-D Euclidean space, i.e., $\mathbb{R}^2$, while for an unmanned aerial vehicle, $\mathscr{P}^i$ is the curve in the 3-D Euclidean space, i.e., $\mathbb{R}^3$. In this paper, we consider the robots in $\mathbb{R}^2$. But, it can be directly extended to $\mathbb{R}^3$.

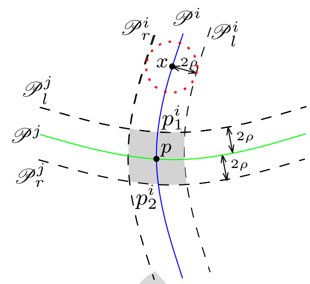


Fig. 1. Safe region of robots in practice. Solid curves $\mathscr{P}^i$ and $\mathscr{P}^j$ are the paths of r_i and r_j. Their safe regions are bounded by the parallel boundaries $\langle \mathscr{P}^i_l, \mathscr{P}^i_r \rangle$ and $\langle \mathscr{P}^j_l, \mathscr{P}^j_r \rangle$. The collision region of r_i around p is the segment $\widehat{p^i_1 p p^i_2}$.

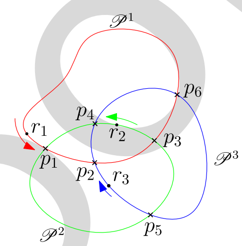


Fig. 2. Three robots are performing persistent motion tasks along fixed paths. The arrows near paths denote the motion directions of the robots.

Definition 2 (Robot Motion): The motion of r_i along $\mathscr{P}^i$ is a binary relation $\rightarrow_{\mathscr{P}^i}$ on $\mathscr{P}^i$, i.e., $\rightarrow_{\mathscr{P}^i} \subset \mathscr{P}^i \times \mathscr{P}^i : \forall x, y \in \mathscr{P}^i$, $(x, y) \in \rightarrow_{\mathscr{P}^i}$, denoted as $x \rightarrow_{\mathscr{P}^i} y$, if r_i can move from x to y along $\mathscr{P}^i$.

The region that may cause collisions between r_i and r_j, denoted as $\mathrm{CR}^{i,j}$, is the intersection of $\mathscr{P}^i$ and $\mathscr{P}^j$, i.e., $\mathrm{CR}^{i,j} = \mathscr{P}^i \cap \mathscr{P}^j$. Thus, r_i's collision region, denoted as CR^i, is defined as the union of $\mathrm{CR}^{i,j}$ for all $j \neq i$, i.e., $\mathrm{CR}^i = \cup_{j \in \mathbb{N}_N \setminus \{i\}} \mathrm{CR}^{i,j}$.

Remark 2: In this paper, each robot is modeled as a mass point theoretically. But in practice, each robot is located by its center and has a safe radius ρ. By safe radius, we mean that the motion region of r_i is the area $\{z \mid \|z - x_i\|_2 < \rho, x_i \in \mathscr{P}^i\}$ and any two robots r_i and r_j should keep a distance 2ρ, i.e., $\|x_i - y_j\|_2 < 2\rho$, where x_i and y_j are their positions. Thus, the safe region of r_i is $P^i_\rho = \{x \mid \|x - x_i\|_2 < 2\rho, x_i \in \mathscr{P}^i\}$. So the practical collision regions are the intersecting parts of the safe regions. For example, as shown in Fig. 1, the area inside the red dotted circle is the safe region of r_i when it is at point x, the blue solid curve is the path of r_i, the pair of dashed curves $\langle \mathscr{P}^i_l, \mathscr{P}^i_r \rangle$ is the boundaries of the practical safe region of r_i. Besides, the intersecting point p represents the gray region.

For example, as shown in Fig. 2, there are three robots r_1, r_2, and r_3, whose paths are $\mathscr{P}^1$ (the red one), $\mathscr{P}^2$ (the green one), and $\mathscr{P}^3$ (the blue one), respectively. The arrows denote their motion directions. The collision set between $\mathscr{P}^1$ and $\mathscr{P}^2$ is $\mathrm{CR}^{1,2} = \mathrm{CR}^{2,1} = \{p_1, p_3\}$, between $\mathscr{P}^2$ and $\mathscr{P}^3$ is $\mathrm{CR}^{2,3} = \mathrm{CR}^{3,2} = \{p_4, p_5\}$, and between $\mathscr{P}^1$ and $\mathscr{P}^3$ is $\mathrm{CR}^{1,3} = \mathrm{CR}^{3,1} = \{p_2, p_6\}$. Hence, their collision sets are $\mathrm{CR}^1 = \{p_1, p_2, p_3, p_6\}$, $\mathrm{CR}^2 = \{p_1, p_3, p_4, p_5\}$, and

$\mathrm{CR}^3 = \{p_2, p_4, p_5, p_6\}$, respectively. Thus, r_1 will collide with r_2 when they are both at p_1 or p_3, and with r_3 when r_1 and r_3 are both at p_2 or p_6. r_2 and r_3 will collide when they are both at p_4 or p_5.

Now we give the persistent motion on which our attention is focused in this paper.

Definition 3 (Persistent Motion): Given a closed path, a robot is doing persistent motion if it can repeatedly traverse the path.

B. Modeling Robot Motion by LTSs

Usually, the path of a robot can be an arbitrary curve such that we cannot give the detailed mathematical formula for this path. This makes it difficult to analyze and design a proper motion controller for the system. Fortunately, discrete representation of robot motion is a well-established method to reduce the computational complexity [29]. Furthermore, it is a common practice in approaches that decompose a control problem into two hierarchies: 1) the high-level discrete planning synthesis and 2) the low-level continuous feedback controller composition [19]. For example, Reveliotis and Roszkowska [30], [31] 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 stages. For their method, each robot should have a global knowledge of the environment. Different with their work, in this paper, we study the motion control problem from the theory of supervisory control of DESs, and discretize the paths directly. Thus, each robot only needs to know its own path, rather than the whole environment. In the sequel, we model the motion of robots by LTSs, based on which we can do further analysis.

Definition 4 [4]: An LTS is a quadruple $\langle S, \Sigma, \rightarrow, s_0\rangle$, where:

1) S is the finite set of states;
2) Σ is the finite set of events;
3) $\rightarrow \subset S \times \Sigma \times S$ is the set of transitions;
4) s_0 is the initial state.

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

Modeling of robot motion contains two stages. The first one is to discretize the paths and the second one is to construct detailed LTSs.

1) Discretization of the Paths: At the first stage, we need to discretize all paths. Consider robot r_i's path $\mathscr{P}^i$.

For any collision region $\mathrm{CR}^{i,j}$, $i, j \in \mathbb{N}_N$, it can be described as a set of disjoint elements, either a segment of a curve or a single point. So the discretization is to abstract each element as a single state. Thus, we can get the discrete form of the collision set between r_i and r_j, denoted as $\mathrm{CS}^{i,j}$. Then the discrete form of the collision set CR^i, denoted as CS^i, can be described as $\mathrm{CS}^i = \cup_{j\in\mathbb{N}_N\setminus\{i\}}\mathrm{CS}^{i,j}$. We call CS^i the set of collision states of r_i. Note that for robots r_i and r_j, they have the same abstracted states corresponding to the collision set $\mathrm{CR}^{i,j}$. For the remaining part of $\mathscr{P}^i$, we use a set of discrete points to partition the path into small subsegments. Each subsegment is abstracted as a discrete state. These states are called private states, denoted as FS^i. Thus, the set of discrete states of $\mathscr{P}^i$, denoted as S^i, is $S^i = \mathrm{FS}^i \cup \mathrm{CS}^i$. S^i is called the state space of r_i.

Remark 3: In practice, we need to consider the safe radius in the discretization process. Thus, though we abstract an intersection point of two paths as a discrete state, this state actually represents a segment of the corresponding path. The practical approach of such abstraction can be described as follows. Suppose $\langle \mathscr{P}_l^j, \mathscr{P}_r^j\rangle$ is r_j's safe region. Thus, r_i's practical collision path with r_j is the set $\mathscr{P}^i \cap \langle \mathscr{P}_l^j, \mathscr{P}_r^j\rangle$. It is a finite set of disjoint segments of $\mathscr{P}^i$. Thus, a state $s_{i,j}$ represents a segment pair $(\mathrm{seg}_i, \mathrm{seg}_j)$ such that $\mathrm{seg}_i^p \subset \mathscr{P}^i \cap \langle \mathscr{P}_l^j, \mathscr{P}_r^j\rangle$, $\mathrm{seg}_j^p \subset \mathscr{P}^j \cap \langle \mathscr{P}_l^i, \mathscr{P}_r^i\rangle$, and $d(\mathrm{seg}_i, \mathrm{seg}_j) < 2\rho$, where $d(\mathrm{seg}_i, \mathrm{seg}_j) = \min\{\|x-y\|_2 | x \in \mathrm{seg}_i, y \in \mathrm{seg}_j\}$. For example, for r_i, the state abstracted from p in Fig. 1 represents the arc $\widehat{p_1^i p p_2^i}$.

From the discretization process, $\mathscr{P}^i$ is divided into a set of segments. We denote each one as $\mathscr{P}_k^i$, $k = 1, 2, \ldots, n^i$, where n^i is the total number of segments. Thus, $\mathscr{P}^i = \cup_{k=1}^{n^i}\mathscr{P}_k^i$, $\mathscr{P}_{k_1}^i \cap \mathscr{P}_{k_2}^i = \emptyset$ for $k_1 \neq k_2$, and $|S^i| = n^i$. Let f^i be the mapping representing the discretization process, i.e., $f^i : \mathscr{P}^i \to S^i$, $\forall \mathscr{P}_k^i$, $k \in \mathbb{N}_{n^i}$, $f^i(\mathscr{P}_k^i) = s_k^i$ if $\mathscr{P}_j^i$ is abstracted as s_j^i in the process of discretization. Moreover, $\forall x \in \mathscr{P}_k^i, f^i(x) = s_k^i$.

According to the process to discretize a multirobot system, we have the following theorem.

Theorem 1: The mapping $f^i : \mathscr{P}^i \to S^i$ satisfies:

1) f^i is a bijection with respect to $\mathscr{P}_k^i, k = 1, 2, \ldots, n^i$;
2) for any point x, $x \in \mathscr{P}^i$, there exists one and only one state $s \in S$ such that $f^i(x) = s$;
3) for any point x, $x \in \mathrm{CR}^{i,j}$, $f^i(x) = f^j(x)$.

Proof: AQ3

1) On one hand, since each $\mathscr{P}_k^i$ is abstracted as a discrete state, there exists a state $s_k^i \in S^i$ such that $f^i(\mathscr{P}_k^i) = s_k^i$. On the other hand, S^i is the set of discrete states abstracted from $\mathscr{P}^i$. Thus, $\forall s_k^i \in S^i$, $\exists \mathscr{P}_k^i$ such that $f^i(\mathscr{P}_k^i) = s_k^i$.
2) $\forall x \in \mathscr{P}^i$, $\exists \mathscr{P}_k^i$ such that $x \in \mathscr{P}_k^i$. From 1), $\exists!\ s_k^i \in S^i$ such that $f^i(x) = s_k^i$.
3) $\forall x \in \mathrm{CR}^{i,j}$, $\exists \mathscr{P}^{i,j} \subset \mathrm{CR}^{i,j}$ such that $x \in \mathscr{P}^{i,j}$. From the discretization, suppose $x \in \mathscr{P}^{i,j}$ is abstracted as $s^{i,j}$. Thus, $f^i(\mathscr{P}^{i,j}) = s^{i,j}$ and $f^j(\mathscr{P}^{i,j}) = s^{i,j}$. Based on 2), $f^i(x) = f^j(x) = s^{i,j}$. ■

From Theorem 1, we can conclude that the process of discretization for a multirobot system does not lose or add any information of the collision locations. Thus, if two robots are in a collision, they are at the same state.

2) LTS Models for Robot Motion: At this stage, we construct the detailed LTS model for each robot motion.

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

Second, consider the set of events. In a multirobot system, each robot can basically either stop at the current state or go to the next state. Thus, we can abstract the event set of r_i as $\Sigma_i = \{\text{move}, \text{stop}\}$.

Third, consider the set of transitions $\rightarrow_i$ for r_i. On one hand, for each state $s_k^i \in S^i$, it is able to 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 $s_{k'}^i$ such that $s_k^i \xrightarrow{\text{move}}_i s_{k'}^i$. This kind of transitions is denoted as $\rightarrow_{i,\text{move}} = \{s_k^i \xrightarrow{\text{move}}_i s_{k'}^i : k = 1, 2, \ldots, n_i$, and $s_{k'}^i$ is uniquely determined by $s_k^i\}$. In fact, the determination of $s_{k'}^i$ can be described as follows. Suppose $f^i(\mathscr{P}_k^i) = s_k^i$. Based on $\mathscr{P}^i$ and the motion direction, we can find $\mathscr{P}_{k'}^i$ such that $\mathscr{P}_{k'}^i$ is the first segment where r_i moves to from $\mathscr{P}_k^i$. Thus, $s_{k'}^i = f^i(\mathscr{P}_{k'}^i)$. Moreover, if r_i moves into $\mathscr{P}_{k'}^i$, the move event is triggered and the transition is fired, and vice verse. On the other hand, robot r_i can stop at any state s_k^i. Thus, there is another transition for each s_k^i, i.e., $s_k^i \xrightarrow{\text{stop}}_i s_k^i$. The set of all this kind of transitions is denoted as $\rightarrow_{i,\text{stop}} = \{s_k^i \xrightarrow{\text{stop}}_i s_k^i : \forall s_k^i \in S^i\}$.

Hence, the detailed LTS model for robot r_i is

$$\mathcal{T}_i = \langle S^i, \Sigma_i = \{\text{move}, \text{stop}\}, \rightarrow_i, s_0^i\rangle \tag{1}$$

where $S^i = \mathrm{CS}^i \cup \mathrm{FS}^i$, $\rightarrow_i = \rightarrow_{i,\text{move}} \cup \rightarrow_{i,\text{stop}}$, and s_0^i is the initial state of r_i.

Based on the construction of the LTS models, we have the following theorem.

Theorem 2: Suppose $\mathscr{P}_{k_1}^i$ and $\mathscr{P}_{k_2}^i$ are two different segments. $\forall x \in \mathscr{P}_{k_1}^i$ and $\forall y \in \mathscr{P}_{k_2}^i$, if $x \rightarrow_{\mathscr{P}^i} y$, then we have $f^i(\mathscr{P}_{k_1}^i) \xrightarrow{\delta}_i f^i(\mathscr{P}_{k_2}^i)$, where δ is a sequence of move and stop events.

Proof: Suppose $f^i(\mathscr{P}_{k_1}^i) = s_{k_1}^i$ and $f^i(\mathscr{P}_{k_2}^i) = s_{k_2}^i$. $\forall x \in \mathscr{P}_{k_1}^i$ and $\forall y \in \mathscr{P}_{k_2}^i$, if it moves from x to y along $\mathscr{P}^i$, r_i traverses a set of pairwise adjacent segments $\mathscr{P}_l^i$, $l = 1, 2, \ldots, L$, obtained from the discretization process. Based on the construction of the move transitions, when a robot traverses from a segment to an adjacent one, the move transition is fired. Thus, when it moves to y through these $\mathscr{P}_l^i$, r_i reaches $s_{k_2}^i$ by firing a set of move transitions. Note that r_i may also stop temporarily at some segments. Thus, $s_{k_1}^i \xrightarrow{\delta}_i s_{k_2}^i$, where δ is a set of move and stop transitions. ■

Theorem 2 states that once a robot moves from one segment to another, the robot described in the LTS model also transits to a corresponding state. Hence, the robots' motion can be described by the constructed LTS models at a higher level.

Let the notation $\bullet_i$ denote r_i's preceding or succeeding operator. Thus, ${}^{\bullet_i}s = \{s' \in S^i | s' \xrightarrow{\text{move}}_i s\}$ and $s^{\bullet_i} = \{s' \in S^i | s \xrightarrow{\text{move}}_i s'\}$. $\forall s \in S^i$, $|{}^{\bullet_i}s| = |s^{\bullet_i}| = 1$. Thus, for convenience, throughout this paper, we directly use the notations ${}^{\bullet_i}s$ and $s^{\bullet_i}$ to denote the unique preceding and succeeding states of s in S^i, respectively. Let s_{cur}^i be the current state of robot r_i.

Clearly, each state in a robot's state space has a self-loop transition; each self-loop transition has a label stop, while other transitions have the label move. For the sake of simplicity, we do not explicitly show the self-loop transitions and labels in the graphic representation of LTS models. At last, we give the LTS description of the whole system.

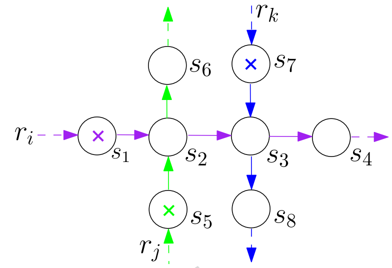


Fig. 3. Part of a multirobot system containing three robots r_i, r_j, and r_k, which are colored purple, green, and blue, respectively. The current states of r_i, r_j, and r_k are s_1, s_5, and s_7, respectively.

Definition 5: Let $\mathcal{T}_i = \langle S^i, \Sigma_i, \rightarrow_i, s_0^i\rangle$ be the LTS model of robot r_i, $i \in \mathbb{N}_N$. The entire system can be described as the parallel composition of all the individual transition systems, i.e., $\mathcal{T} = \mathcal{T}_1 || \cdots || \mathcal{T}_N = \langle C, \Sigma, \rightarrow, c_0\rangle$, where:

1) $C = S^1 \times \ldots \times S^N$;
2) $\Sigma = \cup \Sigma_i$ is the set of labels;
3) $\rightarrow = \cup_{i=1}^N \rightarrow_i$ 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 $j \neq i$;
4) $c_0 = (s_0^1, s_0^2, \ldots, s_0^N)$ is the initial configuration.

For any configuration $c = (s^1, s^2, \ldots, s^N)$, $c(i) = s^i$. The current configuration of the system is denoted as $c_{\text{cur}} = (s_{\text{cur}}^1, s_{\text{cur}}^2, \ldots, s_{\text{cur}}^N)$, and so $c_{\text{cur}}(i) = s_{\text{cur}}^i$.

In the graphic representation of a multirobot system, each circle represents a state and the circle with a colored cross represents the current state of a robot. Arcs with the same color of a cross represent the transitions of the robot represented by this cross. Different colors represent different robots and their transitions. For example, Fig. 3 shows a part of the LTS model of a system containing r_i, r_j, and r_k. The purple cross and arcs represent the current state and the move transitions of r_i, while the green ones represent the current state and move transitions of r_j, and the blue ones the current state and move transitions of r_k. r_i, r_j, and r_k are at s_1, s_5, and s_7, respectively. The transitions of r_j among the given three states are $s_5 \xrightarrow{\text{move}}_j s_2$, $s_2 \xrightarrow{\text{move}}_j s_6$, $s_5 \xrightarrow{\text{stop}}_j s_5$, $s_2 \xrightarrow{\text{stop}}_j s_2$, and $s_6 \xrightarrow{\text{stop}}_j s_6$.

C. Problem Statement

When it is doing persistent motion, a robot may collide with other robots. Moreover, deadlocks among some robots, if not all, may occur and collapse the entire system. Thus, a proper control of the aforementioned system should guarantee that each robot can do persistent motion without causing any collisions or deadlocks with other robots.

By far, we can give the problem statement of the persistent motion control of the multirobot system in terms of LTSs and LTL. It can be described as follows.

Problem: Given the LTS models $\{\mathcal{T}_i\}_{i=1}^N$ of the robots in a system, find a distributed motion controller for the system such that any reachable configuration c satisfies: 1) $\wedge_{i,j \in \mathbb{N}_N} \Box (i \neq j \rightarrow c(i) \neq c(j))$ and 2) $\wedge_{i \in \mathbb{N}_N} \Box (c(i) \rightarrow \Diamond \neg c(i))$.

The first requirement means there are no collisions and the second one means each robot cannot stay at a state forever.

The evolution of a multirobot system relies on a lot of perspectives, such as the motion control algorithms to manage the movement, the sensors to monitor the environment, the communication via a wireless network, and so on. 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 paper, we focus on the design of motion control supervisors. Thus, to simplify the problem, we need some additional assumptions, which nevertheless do not necessarily compromise our technical contributions.

1) *Location and Communication Assumptions:* There are two kinds of ranges for each robot. One is the sensing range and the other is the 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. Thus, we can assume that the communication range is larger than the sensing range. Moreover, we assume that each robot can locate other robots within its sensing range using the sensors, and can communicate with those within the communication range without packet delays, errors, and drops.
2) *Robot Assumptions:* First, each robot can always move along its path with a tolerable derivation. This derivation can be addressed by constraining the robot into the safe radius. Second, different robots have different paths, and each robot knows and only knows its own path in advance.
3) *Path Assumptions:* Each path is a one-way traffic. This means each robot is not allowed to move back. At the initialization stage, each robot has the priori knowledge of its whole path. During the motion, each robot can identify its collision segments on its path via communication before moving into these segments.
4) *System Assumptions:* We regard the multirobot systems as concurrent ones with respect to the high-level abstraction. There are two manifestations of concurrency. For robots without conflicts, they can make decisions and fire transitions automatically; while for robots with conflicts, e.g., requiring the same state to move to, they need to negotiate and determine the robot that can fire the transition. But physically, all robots can move along their continuous paths simultaneously.

IV. Collision Avoidance

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

Definition 6: A multirobot system is in a collision if there exist two robots r_i and r_j, $i \neq j$, such that $s^i_{\text{cur}} = s^j_{\text{cur}}$, where s^i_{cur} and s^j_{cur} are their current states, respectively.

Based on Definition 6, a system is collision-free if and only if $\forall s \in \text{CS}^{i,j}$, there exists at most one robot at s. We assign s a Boolean signal sign_s. When s is empty, $\text{sign}_s = 0$; otherwise, $\text{sign}_s = 1$. A robot can move to s only when s is a private state or $\text{sign}_s = 0$.

Since each robot checks its succeeding state autonomously, there may be several movable robots toward a same empty

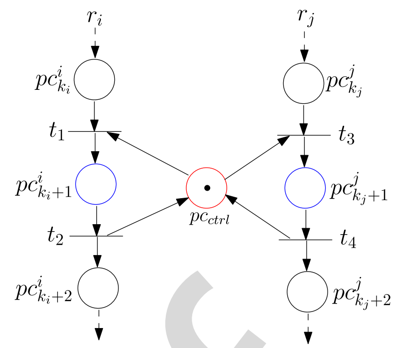


Fig. 4. Petri net model for collision avoidance between r_i and r_j.

state. Thus, they should negotiate with each other to determine which one can actually move forward. There are many negotiation strategies. Since all robots have the same priority, we introduce a simple random selection strategy.

Let *enable* be the set of robots that are able to move into the same crowded region without private states at the current time. The selection can be implemented as follows. Suppose there is a token in this region, and only the robot having this token can move forward. First, a random selection time duration is generated by a robot in *enable* and broadcast to all robots. Second, the token is initially given to an arbitrary robot in *enable*. Third, the token is passed forward to the robots in *enable* during the duration. The rule is that: after it has this token for a well-designed interval, the robot transfers the token to the nearest robot, excluding the robot that just transferred the token. Finally, the robot owning the token at the end of the duration gets the right to move. Once the robot to move forward is determined, *enable* is reset to empty and should be recomputed at the next time. We denote the negotiation process as *Negotiate*(*enable*). It returns the robot to move.

Thus, the collision avoidance framework for r_i is that: after it reaches the preceding state of s, r_i checks the signal sign_s. If $\text{sign}_s = 0$, the negotiation process is executed. If it gets the right to move, r_i moves to s and sign_s is switched to 1; otherwise, it stops at its current state.

We can describe this framework in terms of Petri nets in a more intuitive way. As shown in Fig. 4, 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 $\text{CS}^{i,j}$. 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 $M(pc_{\text{ctrl}}) = 1$, $\text{sign}_s = 0$; otherwise, $\text{sign}_s = 1$. Only when pc_{ctrl} has a token may the transitions t_1 and t_3 be enabled. Indeed, when $M(pc^i_{k_i}) = M(pc^j_{k_j}) = M(pc_{\text{ctrl}}) = 1$, 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., *Negotiate*(*enable*). With this comparison, the negotiation strategies among multiple robots can also be inspired by methods for the selection of firing transitions in Petri nets.

Algorithm 1: Collision Avoidance Algorithm for Robot r_i

Input : $\mathcal{T}_i = \langle S^i, \Sigma_i, \rightarrow_i, s_0^i \rangle$, current state s_{cur}^i, and Sign;
Output: No collision occurs during the motion of r_i;

```
Initialization: s_cur = s^i_cur, s_next = s^{•i}_cur;
if s_next ∈ S^i \ CS^i then
    Execute the transition s_cur --move-->_i s_next;
    if s_cur ∈ CS^i then
        Sign(s_cur) = 0;
    s_cur = s_next; s_next = s^{•i}_cur;
else if Sign(s_next) == 0 then
    Add r_i to enable;
    if Negotiate(enable) == r_i then
        Execute the transition s_cur --move-->_i s_next;
        if s_cur ∈ CS^i then
            Sign(s_cur) = 0;
        Sign(s_next) = 1; s_cur = s_next; s_next = s^{•i}_cur;
else if Sign(s_next) == 1 then
    Stop the motion at the current state;
```

Based on the collision avoidance framework, the distributed algorithm to avoid collisions for robot r_i is shown in Algorithm 1. In the algorithm, Sign is a set of Boolean variables whose elements are sign_s, $s \in \cup_{i \in \mathbb{N}_N} \text{CS}^i$, i.e., $\text{Sign}(s) = \text{sign}_s$. It is a set of public resources, each of which can be broadcast independently to robots. By communicating with some of them, each robot can execute the collision avoidance algorithm in an autonomous way.

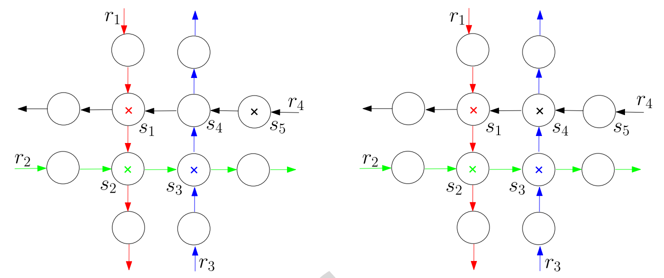
V. Deadlocks and Their Avoidance

In Section IV, we have proposed a distributed algorithm to avoid collisions among multiple robots during their motion. Each robot only checks 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 result.

For example, consider the situation shown in Fig. 5. 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. 5(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 begins to execute its collision avoidance algorithm described in Algorithm 1. Since s_4 is empty, the signal $\text{Sign}(s_4)$ broadcast to r_4 is 0. Hence, the event move in $\mathcal{T}_4$ occurs and causes r_4 to transit to s_4. The system reaches the configuration shown in Fig. 5(b). At this configuration, $r_1 - r_4$ are waiting for the 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.

A. Deadlock Avoidance Algorithm

In this section, we introduce an improved algorithm for the system to avoid both collisions and deadlocks. First, we give the definition and structure properties of deadlocks in the system. Based on the description in [6], we have the following definition.



(a) (b)

Fig. 5. Situation that causes a deadlock among four robots. (a) Before the move of r_4. (b) After the move of r_4.

Definition 7 (Deadlock): A multirobot system is in a deadlock if some of the robots, if not all, are in a circular wait.

Next, we study the properties of deadlocks of the multirobot system in terms of graph theory. For the preliminary knowledge of graph theory, readers can refer to [11].

Definition 8 (Directed Graph): Let $\mathcal{T}_i = \langle S^i, \Sigma_i, \rightarrow_i, s_0^i \rangle$ be the LTS model of robot $r_i, i \in \mathbb{N}_N$. A directed graph of the multirobot system is a two-tuple $G = \langle V, E \rangle$, where:

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

Remark 4:

1) In a directed graph, one of the two endpoints of a directed edge is designated as the tail, while the other endpoint is designated as the head. In an edge, the arrow points from the tail to the head.
2) A directed edge e from v_i to v_{i+1} is denoted as (v_i, v_{i+1}).

Based on the formal modeling of the system, the undirected graph generated from G is a simple graph. Thus, we have the following definitions.

Definition 9 (Cycle): Let $G = \langle V, E \rangle$ be the directed graph of a multirobot system. A cycle of G is a sequence $\langle v_1, e_1, \ldots, v_n, e_n, v_1 \rangle$ such that: 1) $\forall i \in \mathbb{N}_n$, $v_i \in V$, 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) $\forall i_1, i_2 \in \mathbb{N}_n$, $v_{i_1} \neq v_{i_2}$ if $i_1 \neq i_2$; and 3) $\forall j_1, j_2 \in \mathbb{N}_n$, suppose $e_{j_1} \in \rightarrow_{k_1,\text{move}}$ and $e_{j_2} \in \rightarrow_{k_2,\text{move}}$, $k_1 \neq k_2$ if $j_1 \neq j_2$.

For example, as the system shown in Fig. 5, 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., $(s_1, s_2) \in \rightarrow_{1,\text{move}}$, $(s_2, s_3) \in \rightarrow_{2,\text{move}}$, $(s_3, s_4) \in \rightarrow_{3,\text{move}}$, and $(s_4, s_1) \in \rightarrow_{4,\text{move}}$.

In the directed graph of a multirobot 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 Fig. 6(a) and (b). The robots at either configuration are in a cycle. But the robots in Fig. 6(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 10 (Active Edge): Given the graph $\langle V, E \rangle$ of a multirobot system, a directed edge e, $e = (s_1, s_2) \in \rightarrow_{i,\text{move}} \subset E$, is called an active edge if the robot r_i is at s_1.

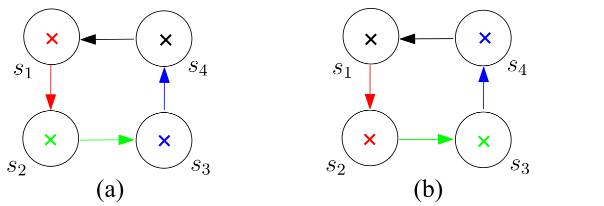


Fig. 6. Two kinds of cycles in a graph. (a) Deadlock cycle. (b) Cycle but not a deadlock cycle.

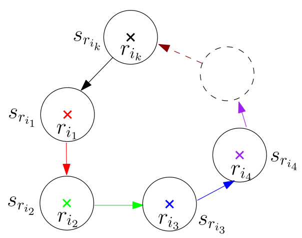


Fig. 7. k robots in a deadlock cycle.

Definition 11 (Deadlock Cycle): A deadlock cycle is a cycle where all edges are active edges.

For example, the four robots in Fig. 6(a) constitute a deadlock cycle since each robot is at the tail of the edge representing one of its transitions. The robots in Fig. 6(b) do not constitute a deadlock cycle although each vertex of the cycle is occupied by a robot.

Theorem 3: A multirobot 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 9 and 11, 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., $e_{i_j} \in \rightarrow_{i_j,\text{move}}$. The cycle is shown in Fig. 7. We can conclude that these k robots are in a circular wait and cannot move any more. Thus, the system is in a deadlock. 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 wait and cannot move anymore.

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 us 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.

Algorithm 2: 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 `moves to` s_{r_i}`; True:` r_i`'s move to` s_{r_i} `can cause a deadlock cycle. */`

Initialization: $q = i$;
while *true* **do**
  `/*` r_q `checks its succeeding state. */`
  $(s_{r_i}, j) = f(p, \rightarrow_q)$;
  if $j == 0$ **then**
    return false;
  else if $s_{cur}^{j}{}^{\bullet j} == s_{r_i}$ **then**
    return true;
  else
    `/* Send the message (`s_{r_i}`,` j`) to` r_j`. */`
    $q = j$;

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 $s_{r_i}^{\bullet i}$. If there exists r_{i_1} such that $s_{\text{cur}}^{i_1} = s_{r_i}^{\bullet i}$, a message is delivered to r_{i_1}. r_{i_1} begins to estimate its succeeding state after receiving the message. If $s_{\text{cur}}^{i_1}{}^{\bullet i}$ is also occupied by a robot, say r_{i_2}, r_{i_2} can receive the corresponding message and begin to estimate the succeeding state. Continue delivering the message until there exists a robot r_{i_k} whose succeeding state either is not occupied by any robots or is s_{r_i}. The former means the transition of r_i to s_{r_i} cannot cause a deadlock, while the latter means there is a deadlock when r_i is at s_{r_i}. The detail is shown in Algorithm 2. In the algorithm $f(s_{r_i}, \rightarrow_j)$ is a function to detect whether the succeeding state of r_j is occupied by a robot. It returns a two-tuple (s_{r_i}, k), where s_{r_i} is a constant state that needs to be checked, and k is the index of the robot that satisfies $s_{\text{cur}}^{k} = s_{\text{cur}}^{j}{}^{\bullet j}$ if $k \neq 0$, whereas $k = 0$ if r_j's succeeding state is not occupied by any robots.

Algorithm 3: Collision and Deadlock Avoidance Algorithm for Robot r_i

Input : The LTS model $\mathcal{T}_i$, current state s_{cur}, and signal $Sign$;
Output: No collisions and deadlocks occur during the motion of r_i;

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

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

Theorem 4: Algorithm 2 can always end by returning a boolean value at any time.

Proof: From the proof of Theorem 3, for any robot r_i, 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 2, each loop is a message delivery. Thus, one of the conditions in lines 4 and 6 of Algorithm 2 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 2, 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 2 only needs to be executed when robot r_i is at a state s satisfying $s^{\bullet i} \in \mathrm{CS}^i$ and $(s^{\bullet i})^{\bullet i} \in \mathrm{CS}^i$. 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 3. Note that since each robot checks deadlock cycles in a distributed way, there may be many robots that can move forward at the same time. Thus, these robots should negotiate with others and only one can move forward because of concurrency.

Now, let us take the system in Fig. 5(a) as an example to explain the distributed execution of Algorithm 3 in a multirobot 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 2 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, $enable = \{r_3\}$. Clearly, $Negotiate(enable) = r_3$. So r_3 moves forward. Thus, with the deadlock avoidance algorithm, the situation shown in Fig. 5(b) cannot occur.

B. 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 $s_{\mathrm{cur}}^{j}{}^{\bullet j} = s$, 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 3.

Proof: Suppose r_i is at s. The satisfaction of the first requirement is directly from Algorithm 1. So we should prove that the second requirement is also satisfied. If r_i can eventually move one step forward, the proposition $s \to \Diamond\neg s$ is satisfied. The arbitrariness of s guarantees that $\Box(s \to \Diamond\neg s)$ 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) $Detect(\mathcal{T}_i, s^{\bullet i}) = 1$ and 2) there exists a robot r_{i_1} such that $s^{\bullet i} = s_{cur}^{i_1}$. We need to prove that r_i can eventually move forward in either situation.

For the first case, there exist a set of robots $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$ such that $s_{\mathrm{cur}}^{i_{j+1}} = s_{\mathrm{cur}}^{i_j}{}^{\bullet i_j}$, $j = 1, 2, \ldots, k-1$, and $s_{\mathrm{cur}}^{i_k}{}^{\bullet i_k} = s^{\bullet i} \triangleq ss$ is empty. Based on the assumption declared above, r_{i_k} can move to ss and then to $ss^{\bullet i_k}$ in the future. When r_{i_k} arrives at $ss^{\bullet i_k}$, $Detect(\mathcal{T}_i, s^{\bullet i}) = 0$ because $s_{\mathrm{cur}}^{i_{k-1}}{}^{\bullet i_{k-1}}$ is now empty. Thus there is no deadlock cycle when r_i is at $s^{\bullet i}$. Hence, r_i can move one step forward.

For the second case, there exist robots $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$ satisfying $s_{\mathrm{cur}}^{i_1} = s^{\bullet i}$ and $s_{\mathrm{cur}}^{i_{j+1}} = s_{\mathrm{cur}}^{i_j}{}^{\bullet i_j}$ for $j = 1, 2, \ldots, k-1$. Moreover, $s_{\mathrm{cur}}^{i_k}{}^{\bullet i_k}$ 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 12 (Admissible Motion): For any robot r_i, the admissible motion is the move that cannot cause any collisions and deadlocks.

Theorem 6 (Maximal Permissiveness): The control policy described by Algorithm 3 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 that are in *enable*. Thus, during the proof of this theorem in terms of r_i, we assume that r_i always wins the negotiation.

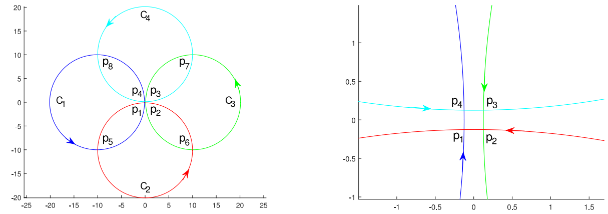
We need to prove that any possible control policies must contain the stopping motion of Algorithm 3. Suppose r_i is at an arbitrary state s at the current moment. On one hand, from the algorithm, r_i will stop its motion in two cases: 1) $Detect(\mathcal{T}_i, s^{\bullet i}) = 1$ (lines 12 and 13) and 2) $s^{\bullet i} \in \mathrm{CS}^i \wedge \mathrm{Sign}(s^{\bullet i}) = 1$ (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 assumption. Thus, for any state s, if r_i stops at s under Algorithm 3, r_i stops at s under any other available control policies. Hence, the proposed algorithm is maximally permissive. ■

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.

VI. Simulation Results and Implementation

A. Simulation Results

In this section, we implement the algorithms in MATLAB. Simulations are carried out for a multirobot system with four robots r_1, r_2, r_3, and r_4, whose paths are shown in Fig. 8. Each path is a circle with a radius of 10 units. Their detailed equations are $C_1 : (x+a)^2+y^2 = 10^2$ (the blue one), $C_2 : x^2+(y+a)^2 = 10^2$ (the red one), $C_3 : (x-a)^2+y^2 = 10^2$ (the green one), and $C_4 : x^2 + (y - a)^2 = 10^2$ (the cyan one), where $a = \sqrt{10^2 - ((\pi/25))^2} + (\pi/25)$. There are totally 8 intersection points, i.e., $p_1 - p_8$.



(a) (b)

Fig. 8. Paths of four robots. $C_1 - C_4$ are the four paths and $p_1 - p_8$ are eight intersection points. (a) Four paths mutually intersecting at $p_1 - p_4$. (b) Magnified view of the enclosed region defined by $p_1 - p_4$.

TABLE I
Polar Coordinates of the Collision Points in Different PCSs

Point	Polar Coordinate System (PCS)			
	ρ_1	ρ_2	ρ_3	ρ_4
p_1	$\frac{499\pi}{250}$	$\frac{126\pi}{250}$	–	–
p_2	–	$\frac{124\pi}{250}$	$\frac{251\pi}{250}$	–
p_3	–	–	$\frac{249\pi}{250}$	$\frac{376\pi}{250}$
p_4	$\frac{\pi}{250}$	–	–	$\frac{374\pi}{250}$
p_5	$\frac{376\pi}{250}$	$\frac{249\pi}{250}$	–	–
p_6	–	$\frac{\pi}{250}$	$\frac{374\pi}{250}$	–
p_7	–	–	$\frac{126\pi}{250}$	$\frac{499\pi}{250}$
p_8	$\frac{124\pi}{250}$	–	–	$\frac{251\pi}{250}$

1. "–" means the point is not in the corresponding path of the PCS. Thus, we needn't to consider its polar coordinates in this PCS.

First of all, for each path C_i, we define a polar coordinate system (PCS), denoted as ρ_i, whose pole and polar axis are, respectively, the center of the path and the ray in the direction of the x-axis, to describe this path. Thus, each point of the path can be expressed by the polar coordinates in the corresponding PCS. For example, each point (x, y) on C_1 can be described by the polar coordinate (r, θ) in ρ_1, such that $x = -a+r\cos\theta$ and $y = r\sin\theta$, where $r = 10$, and $\theta \in [0, 2\pi)$. Since the radial coordinates of all points are equal to 10, we hereby only show the angular coordinate of each point. The angular coordinates of the 8 points in different PCSs are shown in Table I. For example, consider point p_1. p_1 is an intersection point of C_1 and C_2. Thus, its Cartesian coordinate is $(-(\pi/25), -(\pi/25))$. ρ_1 is C_1's PCS, and its pole is $(-a, 0)$. Hence, the polar coordinate of p_1 in ρ_1 is $(10, (499\pi/250))$. Similarly, the polar coordinate of p_1 in ρ_2 is $(10, (126\pi/250))$.

Remark 5: Since all the radial coordinates are equal to 10, each point on a path is uniquely determined by its angular coordinate in the corresponding PCS. Thus, in the rest of this section, the points of a path are described by only the angular coordinates in the corresponding PCS.

TABLE II
DISCRETE POINTS OF THE FOUR PATHS

Path	Angular Coordinates of the Discrete Points
C_1	$\frac{(2k+1)\pi}{250}$: $k = (N^*\backslash\{61, 62, 187, 188\}) \cup \{61.5, 187.5\}$
C_2	$\frac{2k\pi}{250}$: $k = (N^*\backslash\{0, 1, 124, 125\}) \cup \{0.5, 124.5\}$
C_3	$\frac{(2k+1)\pi}{250}$: $k = (N^*\backslash\{62, 63, 186, 187\}) \cup \{62.5, 186.5\}$
C_4	$\frac{2k\pi}{250}$: $k = (N^*\backslash\{0, 125, 126, 249\}) \cup \{125.5, 249.5\}$

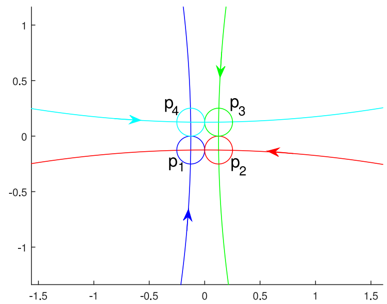


Fig. 9. Deadlock occurs in case 2 under the control of the collision avoidance algorithm.

Each path is discretized into 248 states, which are represented by the discrete points shown in Table II, where $N^* = \{0, 1, 2, \ldots, 249\}$.

We first simulate the motion of the system under the control of Algorithm 1. Consider two different initial configurations of the system.

Case 1: The initial states of $r_1 - r_4$ are $^1(479\pi/250)$, $^2(116\pi/250)$, $^3(229\pi/250)$, and $^4(356\pi/250)$, respectively.

Case 2: The initial states of $r_1 - r_4$ are $^1(479\pi/250)$, $^2(104\pi/250)$, $^3(229\pi/250)$, and $^4(354\pi/250)$, respectively. Here the prefixed superscripts of the angular coordinates denote the indices of the PCSs.

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 move persistently without causing collisions and deadlocks; while with those of case 2, after firing ten transitions simultaneously, they stop at the configuration shown in Fig. 9. 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 1 with Algorithm 3. With this algorithm, the four robots need to negotiate with each other when they want to move to $p_1 - p_4$ simultaneously. Fig. 10 shows 6 snapshots of the simulation.

Suppose the system is now at the configuration shown in Fig. 10(a). At this moment, $r_1 - r_4$ are able to move one step forward based on the condition in line 8 of Algorithm 3. Suppose r_1 wins in the negotiation process, r_1 moves one step forward and reaches p_1. Then, $r_2 - r_4$ and r_1 are able

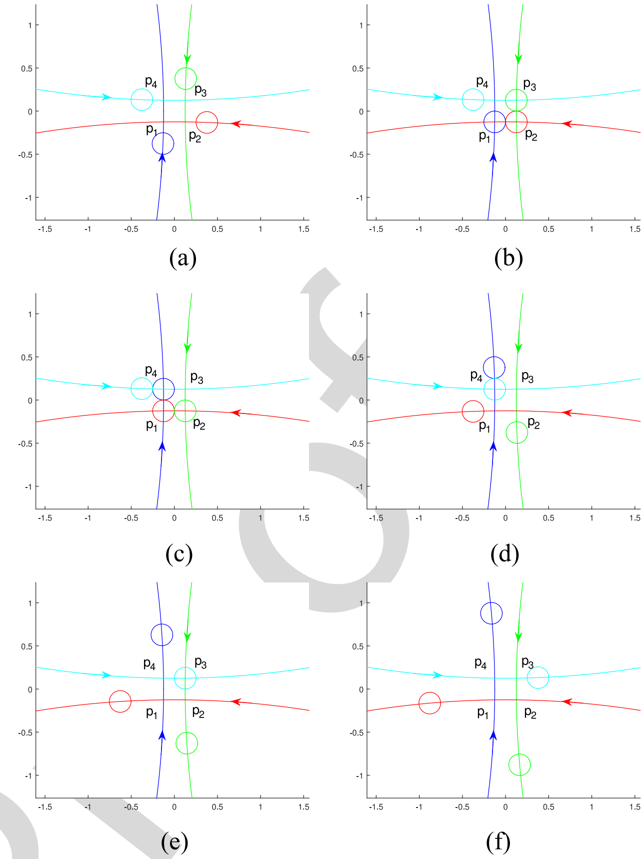


Fig. 10. Six snapshots of the simulation of case 2 under control of deadlock avoidance algorithm. Configurations $c_2 - c_6$ show the process of deadlock avoidance. (a) Configuration c_1. (b) Configuration c_2. (c) Configuration c_3. (d) Configuration c_4. (e) Configuration c_5. (f) Configuration c_6.

to move forward. If r_2 is selected from their negotiation, it moves forward and arrives at p_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 p_3. Therefore, the system reaches the configuration shown in Fig. 10(b). At this configuration, r_4 predicts that its move to p_4 can cause a deadlock cycle $\langle p_1, (p_1, p_2), p_2, (p_2, p_3), p_3, (p_3, p_4), p_4, (p_4, p_1), p_1\rangle$. Hence, r_4 cannot move based on the condition in line 12 of Algorithm 3. Moreover, r_2 and r_3 cannot move forward based on line 21 of their own copy of Algorithm 3. Thus, only r_1 can move one step forward based on line 8 of its Algorithm 3. When r_1 reaches p_4, p_1 is empty. So r_2 is able to move forward and then is selected to move. The move of r_2 releases p_2 such that r_3 is allowed and selected to move to p_2. Thus, the configuration of the system is now shown in Fig. 10(c). At configuration c_3, r_4 cannot move forward since p_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 p_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. 10(d). We can do the similar analysis on how the system reaches the states shown in Fig. 10(e) and (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.

TABLE III
COMPARISON OF THE LENGTH OF THE MAXIMAL EVENT SEQUENCE LEADING A ROBOT TO MOVE 2 CYCLES

Initial Configuration ($\times \frac{\pi}{250}$)	Length of the Maximal Event Sequence		
	Optimal	Soltero's [35]	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

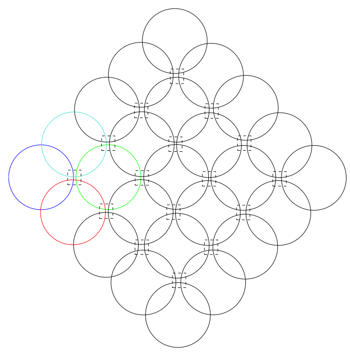


Fig. 11. Extended system from 4 to 25 robots. There exist 16 deadlocks in the system. Each deadlock region is marked by a dashed square.

At last, we give the comparison of the efficiency between our method and that in [35] in terms of the length of event sequences. We study six different initial configurations and count the length of the maximal event sequence which leads a robot to move 2 cycles along its path. The results are shown in Table III. 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 III, our method is an improvement of that in [35].

For a deeper exploration of our algorithm, we further study the systems extended from the original system in Fig. 8(a) by continually adding the deadlock regions $p_1 - p_4$. 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. 11 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 IV. We can find the number of deadlocks increases in propositional 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. 12. Hence, it is important to control a multirobot system with the proposed deadlock avoidance algorithm, which is effective to avoid deadlocks.

TABLE IV
NUMBERS OF ROBOTS AND DIFFERENT DEADLOCKS THAT MAY OCCUR

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

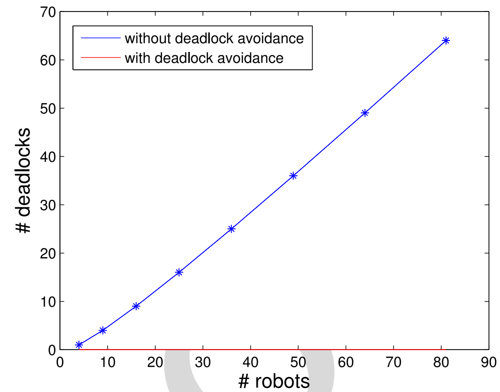


Fig. 12. Number of different deadlocks that may occur in the systems with different robots. Without deadlock avoidance, the number is linearly increased in proportion to the number of robots, while with our deadlock avoidance algorithm, there are no deadlocks during the evolution of the system.

(a) (b)

Fig. 13. Example of autonomous car and the schematic diagram of the crossing. (a) RBCAR from Robotnik. (b) Crossing.

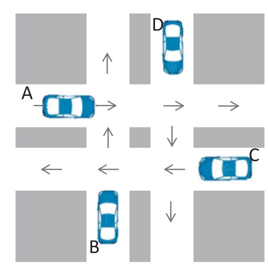


Fig. 14. Four vehicles arrive at the crossing successively.

B. Experimental Implementation

Now we implement our algorithm in a practical scenario where four autonomous vehicles are about to pass through a crossing. Fig. 13(a) shows an example of research autonomous vehicles, and Fig. 13(b) gives the diagram of a crossing.

Let the four vehicles arrive at the crossing successively, as shown in Fig. 14. For convenience, we only give the schematic. Fig. 15 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. 16 shows an intermediate configuration of the system with the collision and deadlock avoidance algorithm

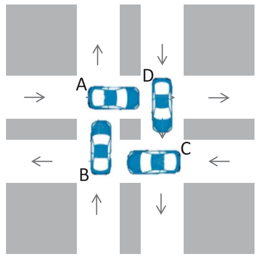


Fig. 15. Four vehicles are in a deadlock.

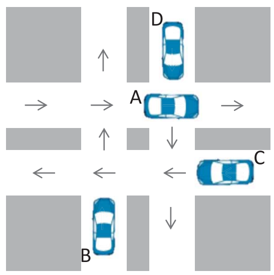


Fig. 16. Intermediate configuration of the motion of the system under the collision avoidance control algorithm in [35].

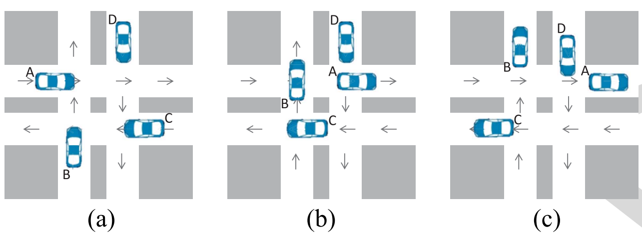


Fig. 17. Three snapshots of the motion of the system under the control of our proposed algorithm. (a) Configuration 1. (b) Configuration 2. (c) Configuration 3.

proposed in [35]. Based on their method, 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 B and C cannot move into the crossing since vehicle A is in the crossing. Fig. 17 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 A, B, and C can be in the crossing at the same time. At configuration 1 in Fig. 17(a), vehicle D cannot move in order to avoid deadlocks, while at configuration 2 in Fig. 17(b), vehicle D cannot move since it is stopped by vehicle A. Only when vehicle A moves away can vehicle D move forward, shown in Fig. 17(c).

VII. Discussion

A. High-Level Discrete Abstraction

Most of the existing work directly studies the motion planning from the kinetics of robots. However, usually the kinetics of a robot is complicated. Thus, it is difficult to plan robot motion, even for a simplified 2-D case [13]. Besides, it is hard to address deadlocks in the multirobot systems from the perspective of the robot kinematics. A common practice in approaches to solving the motion planning is to decompose

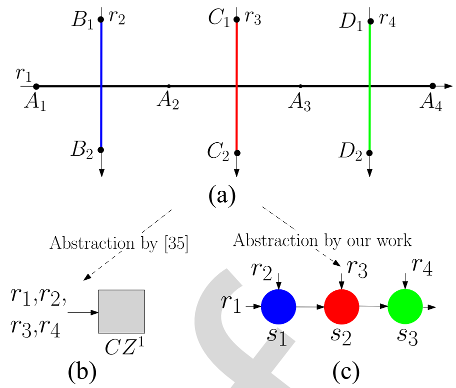


Fig. 18. Different ways to deal with collision regions in [35] and this paper. (a) Paths of four robots. (b) Collision zone. (c) Three different collision states.

the control problem into two hierarchies: 1) the high-level discrete control and 2) the low-level continuous feedback control. In this paper, we study the motion control from the high-level discrete control based on the discrete event systems. Like the work in [35], we abstract the motion of a robot as move and stop. Note that in the high-level, we do not care about the continuous dynamics of robots. Thus, though we simplify the motion control of the multirobot systems, we can make sure that the robots can always avoid unsafe motion, especially the deadlocks, which are hard to avoid during the continuous motion planning. Moreover, such high-level discrete control can also work with the continuous control of the robots.

B. Comparison With Other Work

This paper is an improvement of that in [35]. Soltero *et al.* [35] divided all collision regions into a set of disjoint collision zones. Thus, each robot has at least one collision-free position between any two collision zones. So there do not exist any physical deadlocks. However, this method is too conservative.

For example, as shown in Fig. 18, 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 [35], this region is abstracted as one collision zone CZ^1, shown in Fig. 18(b). When it is in the segment $\widehat{A_1A_4}$, 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 keep 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 starts to move. Thus, to avoid collisions, r_1, r_3, and r_4 need many times of stop.

While with our method, this region is abstracted as three different states $s_1 - s_3$, shown in Fig. 18(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.

VIII. Conclusion

In this paper, we investigate the policy of collision and deadlock avoidance in multirobot systems, where each robot has a predetermined and intersecting path. A distributed algorithm is proposed to avoid collisions and deadlocks. 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.

In the future, we can consider the optimization of the performance of the system by improving the negotiation process, for example, each robot moves as many cycles as possible during a specified time window, or the robots can monitor as much change as possible in the environment. We may also take time into account in the formal models and then propose corresponding motion control algorithms, for example, each robot may be required to stay at different states within different durations. Moreover, in this paper, we suppose the path of each robot is predetermined and only consider the collisions among robots. But the task to generate proper paths for robots to avoid external obstacles is also important but challenging. Thus, one main aspect of the future work is to propose a distributed and fast optimal method to generate a proper path for each robot.

References

[1] J. Alonso-Mora, A. Breitenmoser, P. Beardsley, and R. Siegwart, "Reciprocal collision avoidance for multiple car-like robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, St. Paul, MN, USA, May 2012, pp. 360–366.

[2] S. Akella and S. Hutchinson, "Coordinating the motions of multiple robots with specified trajectories," in *Proc. IEEE Int. Conf. Robot. Autom.*, Washington, DC, USA, May 2002, pp. 624–631.

[3] K. E. Bekris, D. K. Grady, M. Moll, and L. E. Kavraki, "Safe distributed motion coordination for second-order systems with different planning cycles," *Int. J. Robot. Res.*, vol. 31, no. 2, pp. 129–150, Feb. 2012.

[4] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.

[5] L.-W. Chen and P.-C. Chou, "BIG-CCA: Beacon-less, infrastructure-less, and GPS-less cooperative collision avoidance based on vehicular sensor networks," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 11, pp. 1518–1528, Nov. 2016.

[6] E. G. Coffman, M. Elphick, and A. Shoshani, "System deadlocks," *ACM Comput. Surveys*, vol. 3, no. 2, pp. 67–78, Jun. 1971.

[7] M. B. Dias, M. Zinck, R. Zlot, and A. Stentz, "Robust multirobot coordination in dynamic environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, New Orleans, LA, USA, Apr./May 2004, pp. 3435–3442.

[8] H. Fukushima, K. Kon, and F. Matsuno, "Model predictive formation control using branch-and-bound compatible with collision avoidance problems," *IEEE Trans. Robot.*, vol. 29, no. 5, pp. 1308–1317, Oct. 2013.

[9] S. K. Gan, R. Fitch, and S. Sukkarieh, "Real-time decentralized search with inter-agent collision avoidance," in *Proc. IEEE Int. Conf. Robot. Autom.*, St. Paul, MN, USA, May 2012, pp. 504–510.

[10] Y. Guo and L. E. Parker, "A distributed and optimal motion planning approach for multiple mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, Washington, DC, USA, May 2002, pp. 2612–2619.

[11] J. L. Gross, J. Yellen, and P. Zhang, *Handbook of Graph Theory*. Boca Raton, FL, USA: CRC Press, 2013.

[12] 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.

[13] 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,'" *Int. J. Robot. Res.*, vol. 3, no. 4, pp. 76–88, Dec. 1984.

[14] J. Jin, Y.-G. Kim, S.-G. Wee, and N. Gans, "Decentralized cooperative mean approach to collision avoidance for nonholonomic mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, Seattle, WA, USA, May 2015, pp. 35–41.

[15] M. Jäger and B. Nebel, "Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, vol. 3. Maui, HI, USA, Oct. 2001, pp. 1213–1219.

[16] M. Kloetzer and C. Belta, "Temporal logic planning and control of robotic swarms by hierarchical abstractions," *IEEE Trans. Robot.*, vol. 23, no. 2, pp. 320–330, Apr. 2007.

[17] A. Krnjak *et al.*, "Decentralized control of free ranging AGVs in warehouse environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, Seattle, WA, USA, May 2015, pp. 2034–2041.

[18] C. A. Kitts and M. B. Egerstedt, "Design, control, and applications of real-world multirobot systems," *IEEE Robot. Autom. Mag.*, vol. 15, no. 1, p. 8, Mar. 2008.

[19] K. Kim, G. E. Fainekos, and S. Sankaranarayanan, "On the minimal revision problem of specification automata," *Int. J. Robot. Res.*, vol. 34, no. 12, pp. 1515–1535, Aug. 2015.

[20] 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*, A. Koubâa and J. R. M. Dios, Eds. Cham, Switzerland: Springer-Verlag, May 2015, pp. 31–51. AQ4

[21] D. Sun, A. Kleiner, and B. Nebel, "Behavior-based multi-robot collision avoidance," in *Proc. IEEE Int. Conf. Robot. Autom.*, Hong Kong, May/Jun. 2014, pp. 1668–1673.

[22] Z. Li *et al.*, "Trajectory-tracking control of mobile robot systems incorporating neural-dynamic optimized model predictive approach," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 6, pp. 740–749, Jun. 2016.

[23] J. Luo, H. Ni, and M. Zhou, "Control program design for automated guided vehicle systems via Petri nets," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 1, pp. 44–55, Jan. 2015.

[24] D. Morgan, S. J. Chung, and F. Y. Hadaegh, "Model predictive control of swarms of spacecraft using sequential convex programming," *J. Guid. Control Dyn.*, vol. 37, no. 6, pp. 1725–1740, Nov. 2014.

[25] A. Mammeri, D. Zhou, and A. Boukerche, "Animal-vehicle collision mitigation system for automated vehicles," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 9, pp. 1287–1299, Sep. 2016.

[26] E. Olson *et al.*, "Progress toward multi-robot reconnaissance and the MAGIC 2010 competition," *J. Field Robot.*, vol. 29, no. 5, pp. 762–792, Sep. 2012.

[27] A. K. Pamosoaji and K.-S. Hong, "A path-planning algorithm using vector potential functions in triangular regions," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 4, pp. 832–842, Jul. 2013.

[28] P. Rakshit *et al.*, "Realization of an adaptive memetic algorithm using differential evolution and Q-learning: A case study in multirobot path planning," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 4, pp. 814–831, Jul. 2013.

[29] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *J. Field Robot.*, vol. 26, no. 3, pp. 308–333, Mar. 2009.

[30] S. A. Reveliotis and E. Roszkowska, "Conflict resolution in free-ranging multivehicle systems: A resource allocation paradigm," *IEEE Trans. Robot.*, vol. 27, no. 2, pp. 283–296, Apr. 2011.

[31] E. Roszkowska and S. A. Reveliotis, "A distributed protocol for motion coordination in free-range vehicular systems," *Automatica*, vol. 49, no. 6, pp. 1639–1653, Jun. 2013.

[32] 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," *Int. J. Robot. Res.*, vol. 33, no. 12, pp. 1569–1592, Oct. 2014.

[33] 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 *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Chicago, IL, USA, Sep. 2014, pp. 1525–1532.

[34] S. L. Smith, M. Schwager, and D. Rus, "Persistent robotic tasks: Monitoring and sweeping in changing environments," *IEEE Trans. Robot.*, vol. 28, no. 2, pp. 410–426, Apr. 2012.

[35] D. E. Soltero, S. L. Smith, and D. Rus, "Collision avoidance for persistent monitoring in multi-robot systems with intersecting trajectories," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, San Francisco, CA, USA, Sep. 2011, pp. 3645–3652.

[36] X. Wang, M. Kloetzer, C. Mahulea, and M. Silva, "Collision avoidance of mobile robots by using initial time delays," in *Proc. IEEE Conf. Decis. Control*, Osaka, Japan, Dec. 2015, pp. 324–329.

[37] W. Sun, S. Patil, and R. Alterovitz, "High-frequency replanning under uncertainty using parallel sampling-based motion planning," *IEEE Trans. Robot.*, vol. 31, no. 1, pp. 104–116, Feb. 2015.

Yuan Zhou received the M.S. degree in computational mathematics from Zhejiang Sci-Tech University, Hangzhou, China, in 2015. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Nanyang Technological University, Singapore.

He has published several research papers in various journals and conferences, such as the IEEE Transactions on Fuzzy Systems, the IEEE Transactions on Systems, Man, and Cybernetics: Systems, the IEEE Transactions on Automation Science and Engineering, the IEEE Transactions on Reliability, the IEEE International Conference on Automation Science and Engineering, and the International Conference on Software Engineering. His current research interests include robot motion control and planning, multirobot systems, discrete event systems, Petri nets, and system modeling.

Hesuan Hu (M'11–SM'12) received the B.S. degree in computer engineering and the M.S. and Ph.D. degrees in electromechanical engineering from Xidian University, Xi'an, China, in 2003, 2005, and 2010, respectively.

He is currently a Full Professor with Xidian University. His current research interests include discrete event systems and their supervisory control techniques, Petri nets, automated manufacturing systems, multimedia streaming systems, and artificial intelligence. He has over 100 publications in journals, book chapters, and conference proceedings in the above areas.

Dr. Hu was a recipient of many national and international awards including the Franklin V. Taylor Outstanding Paper Award from the IEEE Systems, Man, and Cybernetics Society in 2010 and the Finalists of the Best Automation Paper from the IEEE ICRA Society in 2013 and 2016, respectively. He is an Associate Editor for the IEEE Control Systems Magazine, the IEEE Robotics and Automation Magazine, the IEEE Transactions on Automation Science and Engineering, and the *Journal of Intelligent Manufacturing*. He was on the Editorial Board of over ten international journals.

Yang Liu received the bachelor's and Ph.D. degrees in computer science from the National University of Singapore (NUS), Singapore, in 2005 and 2010, respectively.

He was a Post-Doctoral Fellow with NUS. In 2012, he joined Nanyang Technological University, Singapore, as a Nanyang Assistant Professor. His current research interests include software engineering, formal methods and security, software verification using model checking techniques. This research led to the development of a state-of-the-art model checker, Process Analysis Toolkit.

Zuohua Ding (M'11) received the M.S. degree in computer science and the Ph.D. degree in mathematics from the University of South Florida, Tampa, FL, USA, in 1996 and 1998, respectively.

From 1998 to 2001, he was a Senior Software Engineer with Advanced Fiber Communication, Petaluma, CA, USA. He has been a Research Professor with the National Institute for Systems Test and Productivity, Vail, CO, USA, since 2001. He is currently a Professor and the Director with the Laboratory of Intelligent Computing and Software Engineering, Zhejiang Sci-Tech University, Hangzhou, China. He has authored and co-authored over 70 papers. His current research interests include system modeling, program analysis, service computing, software reliability prediction, and Petri nets.