

# Distributed Motion Control for Multiple Mobile Robots Using Discrete Event Systems and Model Predictive Control

Yuan Zhou, *Member, IEEE*, Hesuan Hu, *Senior Member, IEEE*, Gelei Deng, Kun Cheng, Shang-Wei Lin, Yang Liu, *Senior Member, IEEE*, and Zuohua Ding

**Abstract**—Distributed motion control is critical in multiple mobile robot systems (MMRSs). Current research usually focuses on either discrete approaches, which aim to deal with high-level collisions and deadlocks without considering the low-level motion commands, or continuous approaches, which can optimize low-level continuous commands to mobile robots but cannot deal with deadlocks efficiently. In this paper, by combining discrete and continuous methods, we design a hybrid motion control method for MMRSs where each robot should move along a predefined path. First, each robot’s motion is modeled as a discrete transition system, based on which a real-time supervisory control policy is illustrated to avoid collisions and deadlocks. Second, according to the discrete decisions, the continuous speed at each discrete state is computed using model predictive control and sequential convex programming. The proposed hybrid approach brings two advantages. First, the discrete control component guarantees collision and deadlock avoidance and reduces the scale of the optimization problems. Second, continuous control optimizes the continuous speed in real-time and fulfills other performance requirements like time and energy costs. To move in a fully distributed way, each robot needs to predict the motion of its neighbors by retrieving their immediately available information through communications. The simulation and real-world experimental results show the effectiveness of our approach.

**Index Terms**—Motion planning, multiple mobile robot systems, collision and deadlock avoidance, hybrid control.

## I. INTRODUCTION

A multiple mobile robot system (MMRS) is a system where multiple mobile robots work together to finish given tasks by moving around in a given environment. MMRSs exhibit many advantages, such as increasing spatial coverage and temporal throughput [1] and dramatic capability to resolve task

This work was supported in part by the Ministry of Education, Singapore under its Academic Research Fund (AcRF) Tier 2 (Grant No. MOE-T2EP20120-0004), the National Research Foundation, Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-GC-2023-008), the NRF Investigatorship NRF-NRFI06-2020-0001, the Natural Science Foundation of China (Grant Nos. 61973242, 61573265, 61203037, and 62132014), the New Century Excellent Talents in University (Grant No. NCET-12-0921), and the Major Fundamental Research Program of the Natural Science Foundation of Shaanxi Province (Grant No. 2017ZDJC-34). (*Corresponding author: Hesuan Hu*)

Y. Zhou, G. Deng, S.-W. Lin, and Y. Liu are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: {y.zhou, gelei.deng, shang-wei.lin, yan-liu}@ntu.edu.sg).

H. Hu is with the School of Electro-Mechanical Engineering, Xidian University, Xi'an, Shaanxi 710071, P. R. China (e-mail: hshu@mail.xidian.edu.cn).

K. Cheng is with the State Key Laboratory of Software Development Environment, Beihang University, Beijing, 100191, China (e-mail: chengkun@buaa.edu.cn).

Z. Ding is with the School of Computer Science and Technology, Zhejiang Sci-Tech University, Hangzhou, Zhejiang 310018, China (e-mail: zuohuadong@zstu.edu.cn).

complexity and efficiency [2]. They have been well studied and applied in many fields [2], [3]. Motion planning is one of the most critical tasks in MMRSs, and different approaches have been proposed, which usually focus on unstructured environments [4]–[8].

However, in some scenarios, such as intelligent transportation systems, smart warehouses, and environmental surveillance, robots move in structured environments, i.e., the paths of robots are determined in advance and cannot be changed. For example, autonomous vehicles are required to move along particular roads because of infrastructure limitations; in multi-robot cooperative patrolling, each robot is required to track a set of objects [9]–[11]. In such scenarios, motion planning is to compute a collision-free velocity profile for each robot. Even though off-line methods can be applied to compute velocity profiles for robots in advance, such methods not only require all the information in the system to be predictable, accurate, and determined, but also will compromise the performance and flexibility of the system. While real-time continuous motion planning methods, such as model predictive control (MPC) methods [12], can be applied, deadlocks may arise. Hence, real-time and distributed motion planning methods are investigated for such systems [13]–[16]. In [13], [14], the authors investigate collision avoidance and decision deadlock avoidance in such a system by computing a proper speed for each robot; however, physical deadlocks are not considered. In [15], [16], we consider deadlock and higher-order deadlock avoidance in terms of discrete event control by neglecting the control of continuous velocity. They do not consider the low-level continuous motion control.

Herein, integrating the advantages of discrete and continuous approaches, we develop a real-time, distributed, and hybrid motion planning method, combining discrete event systems and MPC, for robots moving along their predefined and closed paths persistently. By modeling the motion of each robot as a discrete transition system, the method first designs an online supervisory control policy to predict whether the firing of its current transition would cause collisions or deadlocks. Then a continuous control strategy is designed to adjust the motion speed via MPC. One of the key difficulties in this combination lies in coordinating the two methods effectively. Specifically, we need to determine how to translate discrete decisions (i.e., deciding whether the current transition can be fired) into constraints for the optimization problem within the MPC framework. To tackle this challenge, we propose a distributed method that predicts the waiting time of a robot based on the discrete decisions. Subsequently, an optimization problem is formulated by incorporating the motion time constraint,

ensuring that the actual motion time exceeds the predicted waiting time. By leveraging the time constraint, these two methods are effectively coordinated to generate a smooth motion without causing any deadlock or collision. This method can generate optimal speeds for robots such that each one can move to its next state as smoothly as possible. To move in a fully distributed manner, each robot needs to communicate with its neighbors to retrieve their current status for motion prediction. The main contributions of this work are:

- A hybrid and fully distributed approach is proposed for MMRSSs where each robot has a predefined path. It consists of a discrete transition control policy for collision and deadlock avoidance and an MPC-based continuous control module for speed optimization.
- A small and fixed-scale optimization problem is built at each horizon of MPC, being independent of the number of robots in the system. It aims to increase computational efficiency and motion stability.

## II. RELATED WORK

Motion planning is one of the essential issues in MMRSSs, which aims to plan a collision-free motion for each robot. In the past decades, robot motion planning has received considerable attention, deriving various methods [4]–[8], [12], [17]–[24]. They can be roughly divided into discrete and continuous methods.

Usually, discrete methods first discretize the motion space into a set of discrete regions or points and then find a feasible path among them to satisfy some requirements using graph search algorithms, such as A\* and D\*. Some typical discrete methods are formal methods [4], [25], cell decomposition [5], state lattices [6], [26], roadmap methods [7], [17], and incremental sampling-based methods [8], [18]. Discrete methods can simplify motion control; however, they do not explicitly consider the kinematics or dynamics of robots and thus cannot provide low-level continuous control commands, e.g., acceleration or speed, to robots' actuators.

Continuous methods, such as bug-based algorithms [19], potential field-based methods [20], [21], reciprocal velocity obstacles [22], and mathematical programming [12], [23], [24], regard the environment as a continuous Euclidean space and consider the kinematics and/or dynamics of robots. They can generate low-level continuous control inputs to actuators directly. For example, mathematical programming methods consider different constraints, such as mechanism and collision avoidance constraints, and objectives (e.g., motion time and control efforts) to build optimization problems [27]. The optimization problems are usually resolved approximately via convex programming [12], [23]. However, for complex environments or a large number of robots, continuous methods usually cause high computation costs, and they are incapable of dealing with deadlocks.

To leverage their advantages and mitigate their limitations, we propose a hybrid motion planning method for robots performing persistent motion along pre-defined paths. It combines discrete transition systems and model predictive control.

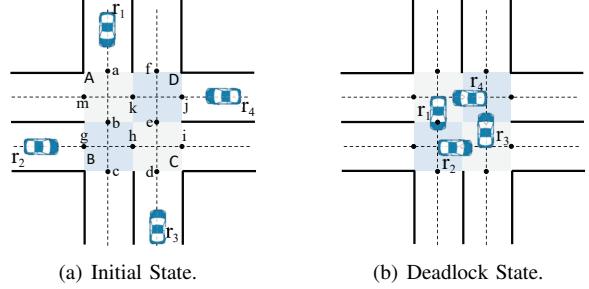


Fig. 1. The occurrence of deadlocks among four robots.

## III. PROBLEM STATEMENT

Suppose that there are  $n$  mobile robots,  $\{r_i : i = 1, 2, \dots, n\}$ , in an MMRSS, and each robot  $r_i$  has a predefined path  $P^i$ . It is a directed curve defined by a parameter equation:  $P^i = P^i(\theta)$ ,  $\theta \in [0, 1]$ , where  $P^i(\theta)$  is a continuously differentiable function mapping from  $[0, 1]$  to the 2D or 3D space.  $d(x, y) = \|x - y\|_2$  denotes the Euclidean distance between the two points  $x$  and  $y$ . Given the  $k$ -th path segment  $P_k^i = \{P^i(\theta) : \theta_1 \leq \theta \leq \theta_2\}$ , the distance between  $x$  and  $P_k^i$ , denoted as  $d(x, P_k^i)$ , can be described as  $d(x, P_k^i) = \inf_{y \in P_k^i} d(x, y)$ .

**Definition 1.** Given a path  $P^i(\theta)$ ,  $p_1 = P^i(\theta_1)$  and  $p_2 = P^i(\theta_2)$  are two arbitrary points on it. The path length from  $p_1$  to  $p_2$ , denoted as  $l^i(p_1, p_2)$ , is the length of the path segment from  $p_1$  to  $p_2$  along the motion direction, which is computed as  $l^i(p_1, p_2) = \int_{\theta_1}^{\theta_2} \|\frac{dP^i(\theta)}{d\theta}\|_2 d\theta$ .

**Definition 2.** Given a robot  $r_i$ , its speed at time  $t$ , denoted as  $v^i(t)$ , can be described as  $v^i(t) = \lim_{\Delta t \rightarrow 0} \frac{l^i(p(t), p(t+\Delta t))}{\Delta t}$ .

For safety reasons, each robot has a safe radius  $\rho$ , and two robots are in a collision if their current positions  $x$  and  $y$  satisfy  $\|x - y\|_2 \leq 2\rho$ . Therefore, the safe region of  $r_i$  can be described as  $\mathbb{A}_{2\rho}^i = \{x | \|x - P^i(\theta)\|_2 \leq 2\rho, \theta \in [0, 1]\}$ . Since each robot is fixed to move along a given path, we can guarantee safety only by controlling the motion speed/acceleration. Moreover, deadlocks are widespread in MMRSSs, especially in robots with predefined paths [28]–[30]. For example, Fig. 1(a) shows four robots crossing an intersection. As given in Fig. 1(b), to avoid collisions, the four robots are in a deadlock and blocked at the intersection forever if they enter the intersection simultaneously without intervention. Hence, the problem studied in this work can be described as follows.

**Problem 1.** Given an MMRSS with predefined paths for robots, determine the real-time motion speeds for robots such that each one can move along its path in a distributed way and guarantee not only collision and deadlock avoidance but also motion smoothness during its motion.

Discrete methods are well-developed in literature to avoid collisions and deadlocks in MMRSSs. However, after high-level abstraction, they cannot optimize the continuous motion of robots. It may cause a robot to make sudden changes in speed and even stop and resume its motion frequently (will give details in our experiments). Such motion is not desired since it is energy-consuming and less comfortable. Hence,

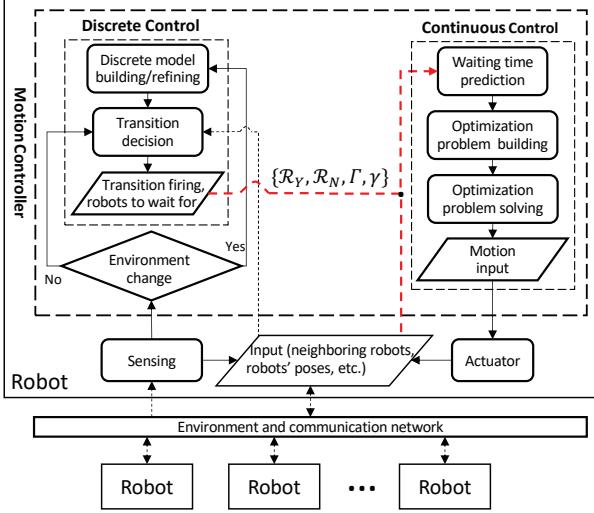


Fig. 2. Framework of the proposed hybrid approach to motion control.

this paper investigates a hybrid approach to robot motion. Based on state transition systems, a discrete control policy is designed to avoid collisions and deadlocks; based on the MPC strategy, an optimal speed control policy is developed to generate continuous motion.

#### IV. DESIGN OF HYBRID APPROACH

The architecture of our proposed hybrid motion controller for each robot is shown in Fig. 2. The motion controller first retrieves information about the environment and the states of its neighbors via its onboard sensors and communication network, then computes the control inputs to the actuator. The computation process contains two phases. (1) Discrete motion control. The robot first builds and refines, if needed, its discrete model based on the path network of the system. Then, by communicating with its neighbors, the robot decides whether the firing of its current transition will cause collisions or deadlocks. The details are given in Section IV-A. (2) Continuous speed optimization. According to the discrete decision of the first phase, the robot computes its minimal motion time at the current state and optimizes its acceleration for the actuator by building and resolving an optimization problem. Section IV-B gives the details. The effectiveness analysis of the proposed approach is given in Section IV-C.

As shown in Fig. 2, the motion accuracy of a robot relies on many aspects, such as the controller, actuator, sensors, and communication network. Since we focus on the design of the motion controller, we assume that other components can always work well. For example, the actuator can respond correctly to its inputs, the sensors can monitor the environment accurately, and the communication network can transmit messages without packet loss and delay. Note, by leveraging the MPC framework, the proposed method can tolerate disturbances in these aspects when the disturbances can be resolved quickly; otherwise, the state-of-the-art technologies [31]–[33] can be integrated into the proposed method.

##### A. Discrete Transition Control

To guarantee the flexibility and efficiency of robot motion, after discretizing a path to discrete states, each time, we only focus on the motion at the current state rather than planning the whole motion along the path. This subsection describes the building of the discrete transition system model and the discrete transition control. In the following subsection, we give continuous motion control. We first give a brief review of the path discretization described in [34].

**Definition 3.** [34] The collision region between  $P^i$  and  $P^j$ , denoted as  $P^{i,j}$ , is a set of segment pairs  $(P_k^i, P_{k'}^j)$  such that  $d(P_k^i, P_{k'}^j) = \inf_{x \in P_k^i} d(x, P_{k'}^j) \leq 2\rho$ , where  $P_k^i$  and  $P_{k'}^j$  are the  $k$ -th and  $k'$ -th maximal continuous segments in  $P^i \cap \mathbb{A}_{2\rho}^j$  and  $P^j \cap \mathbb{A}_{2\rho}^i$ , respectively,  $k, k' \in \mathbb{N}^+$ , and  $\mathbb{N}^+$  is the set of positive integers.

Hence, the collision segments of  $r_i$  can be described as  $P^i = \bigcup_j \bigcup_k \{P_k^i : \exists P_{k'}^j, \text{ such that } (P_k^i, P_{k'}^j) \in P^{i,j}\}$ . It is a set of non-overlapping segments by merging the overlapping ones. The rest of  $P^i$  is a set of collision-free segments, called private segments. All collision and private segments form a partition of  $P^i$ . An example of the discretization process can be found in the Appendix and [34].

Each collision segment is abstracted to a discrete collision state. Two collision segments belonging to two different robots are abstracted to the same collision state if the two robots may cause collisions at these two segments, respectively. Each collision-free segment can be further divided into several shorter sub-segments based on, e.g., the sensing range. In this way, collision-free segments are abstracted to a set of private states. Note that each robot at a state can retrieve the path segment represented by this state. Let  $S_\alpha^i$  and  $S_\beta^i$  be the sets of collision and private states, respectively. Then,  $S^i = S_\alpha^i \cup S_\beta^i$ . Given a state  $s \in S^i$ , its represented path segment is denoted as  $P_s^i$ , and the tail and head of  $P_s^i$  are denoted as  $A_s^i$  (called the start of  $s$ ) and  $B_s^i$  (called the end of  $s$ ), respectively, where robot  $r_i$  moves from  $A_s^i$  to  $B_s^i$  at  $s$ . The length of  $P_s^i$ , i.e.,  $l^i(A_s^i, B_s^i)$ , is denoted as  $l_s^i$ . The set of transitions among the discrete states, denoted as  $T^i$ , is determined as follows: Given two states  $s_1, s_2 \in S^i$ ,  $(s_1, s_2) \in T^i$  if  $B_{s_1}^i = A_{s_2}^i$ . Hence, the transition system for  $r_i$  is  $\mathcal{T}^i = \langle S^i, T^i \rangle$ . According to the discretization process, for any two robots  $r_i$  and  $r_j$  whose state spaces are  $S^i$  and  $S^j$  respectively, there are no collision states such that they are consecutive states in  $S^i$  and  $S^j$  simultaneously. Hence, our discretization will not cause deadlocks between two robots. However, to guarantee maximal permissive motion, some refined discretization processes may generate such consecutive states (refer to the supplementary file for a detailed example). This may increase control complexity and will be investigated in future work.

Given the transition system  $\mathcal{T}^i$ ,  $\forall s \in S^i$ ,  $Pre_i(s)$  and  $Pos_i(s)$  denote the preceding and succeeding states of  $s$  in  $S^i$ , respectively, i.e.,  $(Pre_i(s), s) \in T^i$  and  $(s, Pos_i(s)) \in T^i$ . Let  $s_c^i$  be the current state of  $r_i$ . Then,  $(s_c^i, Pos_i(s_c^i))$  is the current transition of  $r_i$ . At the discrete control phase,  $r_i$  determines whether its current transition can be fired without causing

collisions or deadlocks. In terms of the discrete model,  $r_i$  at  $s$  is in a *collision* if there is another robot at  $s$  simultaneously, while it is in a *deadlock* if there are a set of robots  $r_1, \dots, r_k$  such that  $\text{Pos}_i(s) = s_c^1$ ,  $\text{Pos}_j(s_c^j) = s_c^{j+1}$  for  $j = 1, \dots, k-1$ , and  $\text{Pos}_k(s_c^k) = s$ , where  $s_c^j$  is the current state of  $r_j$ .

**Definition 4.** Suppose that  $r_i$  is at  $s$ . The transition  $(s, \text{Pos}_i(s))$  can be enabled if there are no collisions or deadlocks when  $r_i$  is at  $\text{Pos}_i(s)$ . Firing of its current transition  $(s, \text{Pos}_i(s))$  transits  $r_i$  to  $\text{Pos}_i(s)$ .

Based on the above definition, a robot is movable if its current transition is enabled, and it cannot transit to its next state until its current transition can be fired. Only when its next state is a collision state may a robot collide with others. Hence, at any time instant, to avoid collisions, each robot needs to retrieve the status of its collision states within its sensing range. For each robot, the status of its collision states can be described by a set of Boolean variables. Let  $\omega^i$ ,  $\omega^i = \{\omega^i(s) : s \in S_\alpha^i\}$ , be the set of such Boolean variables:  $\omega^i(s) = 1$  if  $s$  is occupied by other robots; otherwise,  $\omega^i(s) = 0$ . Thus, the collision avoidance strategy is that: If the Boolean variable related to the next state is 1, then the transition cannot be enabled.

Besides collision avoidance, a robot needs to retrieve the current states of its neighbors via a multi-hop communication path to avoid deadlocks. Suppose that  $r_i$  is at  $s$  and  $\omega^i(\text{Pos}_i(s)) = 0$ . To check whether  $(s, \text{Pos}_i(s))$  can be enabled,  $r_i$  needs to check further whether its stay at  $\text{Pos}_i(s)$  would cause any deadlock. The procedure can be concisely summarized as follows. For more details, please refer to [15]. First,  $r_i$  retrieves the status of  $\tilde{s}^i = \text{Pos}_i(\text{Pos}_i(s))$ . If another robot, say  $r_{j_1}$ , is at  $\tilde{s}^i$ , then  $r_{j_1}$  is activated to check the status of  $\text{Pos}_{j_1}(\tilde{s}^i)$ , denoted as  $\tilde{s}^{j_1}$ . Similarly, if  $\tilde{s}^{j_1}$  is also occupied by  $r_{j_2}$ ,  $r_{j_2}$  starts to check the status of  $\text{Pos}_{j_2}(\tilde{s}^{j_1})$ . The procedure is terminated when there exists a robot whose next state is  $\text{Pos}_i(s)$  or is neither  $\text{Pos}_i(s)$  nor occupied. The former means that there exists a deadlock, and  $(s, \text{Pos}_i(s))$  cannot be enabled, while the latter means that  $(s, \text{Pos}_i(s))$  can be enabled. We denote this deadlock detection process as  $\mathcal{D}(r_i, \text{Pos}_i(s))$ .  $\mathcal{D}(r_i, \text{Pos}_i(s)) = 0$  means that no deadlocks are generated if  $r_i$  moves to  $\text{Pos}_i(s)$ , while  $\mathcal{D}(r_i, \text{Pos}_i(s)) = r_j$  means that a deadlock occurs if  $r_i$  is at  $\text{Pos}_i(s)$ , and  $r_j$  is the last one in the iteration, i.e.,  $\text{Pos}_j(s_c^j) = \text{Pos}_i(s)$ , where  $s_c^j$  is the current state of  $r_j$ . In this case,  $r_i$  needs to wait for  $r_j$  to move away from  $\text{Pos}_i(s)$ . Note that in this paper, to simplify the description, we do not consider higher-order deadlocks described in [16]. However, we can directly replace  $\mathcal{D}(r_i, \text{Pos}_i(s))$  by the procedure of higher-order deadlock prediction given in [16].

Since the robots detect collisions and deadlocks in a distributed manner, they may make decisions simultaneously and result in conflicts, i.e., simultaneous firings of enabled transitions result in a collision or deadlock. Hence, an enabled transition sometimes cannot be fired. Indeed, these robots need to negotiate with each other to determine whose transitions can be fired. Before giving the negotiation process, we first introduce the hybrid state of a robot.

---

### Algorithm 1: Negotiation among the robots in $\mathcal{R}_X$ .

---

```

Input : Movable robots  $\mathcal{R}_X$ , and their current hybrid states  $(s^i, p^i, v^i, l^i)$  and signals  $\{\omega^i\}$ .
Output:  $\mathcal{R}_Y$ : the set of robots that can fire their current transitions;  $\mathcal{R}_N$ : the set of robots that cannot fire their current transitions;  $\Gamma = \{(r_i, r_j, t_w(i, j)) : i \in \mathcal{R}_N\}$ :  $r_i$  needs to wait for  $r_j$  for a time duration  $t_w(i, j)$ .
1 Initialization:  $\nu = \{\nu^i = \omega^i : \forall i \in \mathcal{R}_X\}; \mathcal{R}_Y = \emptyset; \mathcal{R}_N = \emptyset; \Gamma = \emptyset;$ 
2 Compute time to the next state:  $t^i = (l_{s^i}^i - l^i)/v^i, \forall r_i \in \mathcal{R}_X;$ 
3 while  $\mathcal{R}_X \neq \emptyset$  do
4    $r_k = \arg \min_{r_i \in \mathcal{R}_X} t^i; s_x = \text{Pos}_k(s^k);$ 
5   if  $\exists r_j \in \mathcal{R}_X$  such that  $\nu^j(s_x) = 1 \parallel \mathcal{D}(r_k, s_x) = r_j$  based on  $\nu$  then
6      $\mathcal{R}_N = \mathcal{R}_N \cup \{r_k\};$ 
7      $t_w(k, j) = l^j(p^j, B_{s_x}^j)/v^j;$ 
8      $\Gamma = \Gamma \cup \{(r_k, r_j, t_w(k, j))\};$ 
9   else
10     $\mathcal{R}_Y = \mathcal{R}_Y \cup \{r_k\}; \nu^k(s_x) = 1;$ 
11    $\mathcal{R}_X = \mathcal{R}_X \setminus \{r_k\};$ 
12 return  $\{\mathcal{R}_Y, \mathcal{R}_N, \Gamma\}$ 

```

---

**Definition 5.** The hybrid state of a robot  $r_i$  at time  $t$  is a quadruple  $(s^i(t), p^i(t), v^i(t), l^i(t))$ , where  $s^i(t) \in S^i$  and  $p^i(t) \in P_{s^i(t)}^i$  are the state and position of  $r_i$  at time  $t$ , respectively,  $v^i(t)$  is the speed, and  $l^i(t) = l^i(A_{s^i(t)}^i, p^i(t))$  is the path length that  $r_i$  has moved at  $s^i(t)$ .

Suppose that  $X$  is a collision region that may exhibit collisions or deadlocks, and  $\mathcal{R}_X$  is the set of robots that are moving into/in  $X$  at the current instant. All robots in  $\mathcal{R}_X$  should negotiate with each other to decide which robots can fire their current transitions. Here we introduce a heuristic negotiation strategy. The main idea is that the earlier the robot arrives at its next state, the higher priority the robot possesses to make a decision, while the remaining robots make their decisions based on the decisions made by the previous ones. Algorithm 1 shows the negotiation process. First, each robot predicts and broadcasts its time to arrive at its next state (Line 2). Then, one by one, each robot in  $\mathcal{R}_X$  decides whether it can fire its current transition (Lines 3–10) based on the temporary Boolean variables  $\nu = \{\nu^i : r_i \in \mathcal{R}_X\}$ , where  $\nu^i$  is initialized to  $\omega^i$ . Suppose that among the remaining robots in  $\mathcal{R}_X$ ,  $r_k$  is the robot with the shortest arriving time to its next state. Based on the Boolean variables  $\nu$ ,  $r_k$  checks whether its next state  $s_x$  is temporarily occupied or whether there exists a deadlock if it is at  $s_x$ . If one of the two conditions is satisfied,  $r_k$  cannot fire its current transition. So  $r_k$  needs to determine its waiting relation  $(r_k, r_j, t_w(k, j))$ , including the robot to directly wait for (i.e.,  $r_j$ ) and the possible waiting time (i.e.,  $t_w(k, j)$ ) (Lines 5–8). Otherwise,  $r_k$  can fire its current transition, and the corresponding temporary variable  $\nu^k(s_x)$  is changed to 1 (Line 10). At last,  $r_k$  is removed from  $\mathcal{R}_X$ . The negotiation process returns the sets of robots that can and cannot fire their current transitions, i.e.,  $\mathcal{R}_Y$  and  $\mathcal{R}_N$ , respectively, and the set of waiting relations, i.e.,  $\Gamma$ , of the robots in  $\mathcal{R}_N$ .

---

**Algorithm 2:** Decision making on the firing of the current transition of  $r_i$ .

---

**Input :** Transition system  $\mathcal{T}^i$ , the current state  $s$ , and collision region  $X$  containing  $Pos_i(s)$ .

**Output:**  $\mathcal{R}_Y$ : The set of robots that can fire their current transitions;  $\mathcal{R}_N$ : The set of robots that cannot fire their current transitions;  $\Gamma = \{r_k, r_j, t_w(k, j)\}$ : The set of waiting relation between two robots and the waiting time;  $\gamma$ : The decision made by  $r_i$ .

```

1  $\mathcal{R}_Y = \emptyset, \mathcal{R}_N = \emptyset, \Gamma = \emptyset;$ 
2 if  $Pos_i(s) \in S_\beta^i$  then
3    $\gamma = 2;$ 
4 else if  $\omega^i(Pos_i(s)) = 1$  then
5    $\gamma = 0;$ 
6 else
7   if  $\mathcal{D}(r_i, Pos_i(s)) \neq 0$  then
8      $\gamma = 0;$ 
9   else
10    ( $s, Pos_i(s)$ ) is enabled and add  $r_i$  to  $\mathcal{R}_X$ ;
11     $\{\mathcal{R}_Y, \mathcal{R}_N, \Gamma\}$  = Algorithm 1;
12    if  $r_i \in \mathcal{R}_Y$  then
13       $\gamma = 2$ 
14    else
15       $\gamma = 1$ 
16 return  $\{\mathcal{R}_Y, \mathcal{R}_N, \Gamma, \gamma\}.$ 

```

---

Algorithm 2 shows the decision-making on whether  $r_i$  can fire its current transition. It consists of four cases. (i) Leave to a private state:  $r_i$  can always fire its current transition to a private state (Lines 2 and 3); (ii) Collisions: the current transition of  $r_i$  cannot be enabled when another robot is at the next state (Lines 4 and 5); (iii) Deadlocks:  $r_i$  cannot enable its current transition if a deadlock is detected (Lines 7 and 8); (iv) Negotiation: If  $r_i$  can move to a collision state, it negotiates with its neighbors based on Algorithm 1 to decide whether the current transition can be fired (Lines 10–15). Note that  $\gamma = 0$  means that the robot cannot move one step forward due to robot collisions or system deadlocks,  $\gamma = 1$  means that the robot is movable but cannot move forward due to the loss of the negotiation process, and  $\gamma = 2$  means that the robot can move one step forward.

Once a robot checks that it cannot fire its current transition, i.e., Algorithm 2 outputs  $\gamma = 0$  or  $\gamma = 1$ , the robot needs to optimize its speed such that no collisions or deadlocks are generated when it transits to the next state.

### B. Continuous Speed Adjustment

This subsection describes the MPC-based method to optimize the speed of a robot if it cannot fire its current transition. Once a robot transits to the next state, it has to re-compute its speed. It means that the motion derived from the previous state is not feasible anymore. Hence, at any time instant, the prediction horizon is the path from the current position to the end of the path segment at the current state. Since the motion time at each state is varied case by case, we consider the path-horizon MPC process rather than following the conventional time-horizon MPC strategies. Hence, the tuning parameters are the control horizon and the path-length step. In this work,

---

**Algorithm 3:** Computation of  $r_i$ 's waiting time based on its negotiation process.

---

**Input :**  $\mathcal{R}_Y, \mathcal{R}_N$ , and  $\Gamma$  from Algorithm 1.

**Output:**  $t_w(i)$ .

```

1  $\Gamma_i = (r_i, r_j, t_w(i, j)); t_w(i) = t_w(i, j);$ 
2 while  $r_j \in \mathcal{R}_N$  do
3   Retrieve  $\Gamma_j = (r_j, r_k, t_w(j, k));$ 
4    $t_w(i) = \max\{t_w(i), t_w(j, k)\};$ 
5    $r_j = r_k;$ 
6 return  $t_w(i).$ 

```

---

we apply the default one-step control horizon, and the path length of each step is determined such that the robot can fully stop at each step. Even though different tuning methods have been proposed in the literature [35]–[37] to optimize the tuning parameters, it is beyond the scope of this paper and will be future work.

At any time  $t$ , the state and position of  $r_i$  are denoted as  $s^i(t)$  and  $p^i(t)$ , respectively; the path length from the tail  $A_{s^i(t)}^i$  to  $p^i(t)$  is  $l^i(t) = l^i(A_{s^i(t)}^i, p^i(t))$ ; the speed and acceleration of  $r_i$  at  $t$  are denoted as  $v^i(l^i(t))$  and  $a^i(l^i(t))$ , respectively. The motion time from  $p^i(t)$  to  $B_{s^i(t)}^i$  is denoted as  $t^i(l^i(t), l_{s^i(t)}^i)$ . Recall that  $l_{s^i(t)}^i$  is the length of  $P^i$  at  $s^i(t)$ . Without ambiguity, we omit the time parameter  $t$  in the following description. Given the current time instant  $t_c$ , the current hybrid state of  $r_i$  can be described as  $(s, p^i(t_c), v^i(l_c), l_c)$ , where  $l_c = l^i(t_c)$ . In the sequel, we formulate the optimization problem in the MPC process of  $r_i$  at  $t_c$ .

First, the kinematic equations can be described as follows.

$$t^i(l_c, l_s^i) = \int_{l_c}^{l_s^i} \frac{1}{v^i(l)} dl, \quad (1)$$

$$\frac{1}{2}v^i(l_s^i)^2 - \frac{1}{2}v^i(l_c)^2 = \int_{l_c}^{l_s^i} a^i(l) dl. \quad (2)$$

Second, an optimization problem is generated based on the discrete decision and kinematic equations. As described above, a robot  $r_i$  may not be able to fire its current transitions due to two situations: (1) the negotiation process forbids the robot to fire its current transition, and (2) collisions or deadlocks occur after the firing of the current transition. Once  $r_i$  decides that it cannot fire its current transition, it needs to determine the minimal motion time at the current state.

Given the negotiation results, we first compute the minimal motion time under the first situation. From the negotiation process,  $r_i$  can retrieve the minimal sequence of waiting robots, which satisfies: 1) the former robots need to wait for the latter ones, 2) all robots, except the last one, belong to  $\mathcal{R}_N$ , and 3) the last robot belongs to  $\mathcal{R}_Y$ . Based on this sequence,  $r_i$  can compute its minimal motion time. The computation procedure is shown in Algorithm 3. According to  $(r_i, r_j, t_w(i, j))$ ,  $r_i$  decides that it needs to wait for  $r_j$  for a time duration  $t_w(i, j)$  (Line 1). Hence,  $r_i$  can retrieve  $\Gamma_j$  from  $\Gamma$ , based on which  $r_i$  further determines the robot, alias  $r_k$ , as well as the time duration  $t_w(j, k)$ , that  $r_j$  needs to wait for (Line 3). Then,  $r_i$  updates its waiting time (Line 4) and sets  $r_j$  to  $r_k$  for the next iteration (Line 5). The procedure iterates until  $r_i$  finds a robot belonging to  $\mathcal{R}_Y$ , i.e., a robot that can fire its current transition.

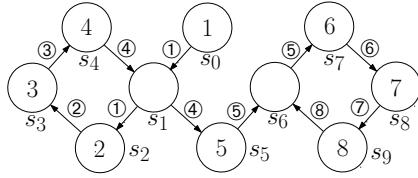


Fig. 3. Illustration of enable-dependent robots. The arrow with  $\textcircled{1}$  denotes the transition of  $r_n$ .

In the sequel, we compute the minimal motion time in the second situation.

**Definition 6.** Suppose that  $r_i$  is required to move to  $s$ . We say  $r_i$  needs to wait for the move of  $r_j$  directly if  $s$  is occupied by  $r_j$  or  $\mathcal{D}(r_i, s) = r_j$ . Such a direct waiting relation is denoted as  $r_i \preceq r_j$ .

**Definition 7.** A robot  $r_j$  is an enable-dependent robot of  $r_i$  at state  $s$  if  $r_i \preceq r_j$  or there exist a sequence of robots  $r_{i_1}, \dots, r_{i_j}$  such that  $r_i \preceq r_{i_1} \preceq \dots \preceq r_{i_j} \preceq r_j$ . The set of all enable-dependent robots is denoted as  $\mathcal{R}^i(s)$ .

$r_i$  can determine the set of enable-dependent robots  $\mathcal{R}^i(s)$  in a distributed way via a multi-hop communication path. First,  $r_i$  can determine its direct waiting relation by checking the status of  $Pos_i(s)$  and  $\mathcal{D}(r_i, Pos_i(s))$ . If  $r_i$  detects that  $Pos_i(s)$  is occupied by another robot  $r_{i_1}$  or  $\mathcal{D}(r_i, Pos_i(s)) = r_{i_1}$ , i.e.,  $r_i \preceq r_{i_1}$ ,  $r_i$  sends a message  $(r_i, Pos_i(s))$  to  $r_{i_1}$  to inform that  $r_{i_1}$  needs to move to  $Pos_{i_1}(Pos_i(s)) \triangleq \tilde{s}^{i_1}$ . When  $r_{i_1}$  receives the message from  $r_i$ ,  $r_{i_1}$  starts to determine its direct waiting relation by checking the status of  $\tilde{s}^{i_1}$  or deriving  $\mathcal{D}(r_{i_1}, \tilde{s}^{i_1})$ . Similarly,  $r_{i_1}$  can retrieve its direct waiting relation, say  $r_{i_1} \preceq r_{i_2}$ , and send a notification message to  $r_{i_2}$ . So  $r_{i_2}$  can begin to retrieve its direct waiting relation. By iterating over all the robots, the process stops when a robot, say  $r_{i_j}$ , detects that it can enable all the transitions to the required state. In this way,  $r_i$  can retrieve its enable-dependent robots at  $s$ , i.e.,  $\mathcal{R}^i(s) = \{r_{i_1}, \dots, r_{i_j}\}$ . Each robot will also send back its current speed and the path length required to move during the communication, so  $r_i$  can predict its minimal motion time at  $s$ .

For example, as shown in Fig. 3,  $r_1$  is at  $s_0$  currently. By deriving  $\mathcal{D}(r_1, s_1)$ , a deadlock would be identified if  $r_1$  moved to  $s_1$ , so  $r_1$  retrieves a direct waiting relation  $r_1 \preceq r_4$ . Hence,  $r_1$  sends a message to  $r_4$ , and  $r_4$  needs to retrieve its direct waiting relation with respect to  $s_5$ . After checking the status of  $s_5$ ,  $r_4$  detects that  $r_5$  is at  $s_5$  and sends a message to  $r_5$ . After receiving the message,  $r_5$  finds that it needs to move to  $s_6$ . So  $r_5$  detects whether it is movable to  $s_6$ . Since  $s_6$  is not occupied by other robots,  $r_5$  further derives  $\mathcal{D}(r_5, s_6)$ . Since  $\mathcal{D}(r_5, s_6) = r_8$ ,  $r_5$  sends a message to  $r_8$ , and  $r_8$  needs to check whether it can move to the next state of  $s_6$ , i.e., checking the status of  $Pos_8(s_6)$ . Assume that  $Pos_8(s_6)$  is a private state. Hence,  $r_8$  finally sends back a message, including its speed and path length to move, to  $r_5$ . Consequently,  $r_5$  will send back to  $r_4$  this information by adding its speed and path length to move. Similarly,  $r_4$  adds its speed and the path length to move to the message and sends it to  $r_1$ . Via such a multi-hop communication path,  $r_1$  retrieves its enable-dependent robots  $\mathcal{R}^1(s_0) = \{r_4, r_5, r_8\}$  and their speeds and path lengths needed to move.

Based on the definition of enable-dependent robots,  $r_i$  cannot leave  $s$  until all robots in  $\mathcal{R}^i(s)$  arrive at the required states. For each robot  $r_j \in \mathcal{R}^i(s)$ , its current position and speed are  $p^j$  and  $v^j$ , respectively. During the enable-dependent retrieval process,  $r_j$  determines that it needs to move to  $\tilde{s}^j$ , which means that  $r_j$  is required to move to at least  $A_{\tilde{s}^j}^j$ . Hence, the path length that  $r_j$  needs to move, denoted as  $l^j(i)$ , is  $l^j(p^j, A_{\tilde{s}^j}^j)$ , and the predicted motion time is  $t_p^i(j) = l^j(i)/v^j$ . In the sequence of enable-dependent robots, the last robot, say  $r_k$ , is a movable robot whose minimal motion time  $t_w(k)$  can be computed by Algorithm 3. Hence,  $r_i$ 's minimal motion time at  $s$  can be computed as:

$$t_w(i) = \max\{\max_{r_j \in \mathcal{R}^i(s)} t_p^i(j), t_w(k)\} \quad (3)$$

Hence, the local optimization problem for  $r_i$  to compute its optimal speed is shown in (4) in terms of (4a)–(4e). First, the objective function is given in (4a). It consists of two targets: motion smoothness, which means that the robot is expected to move as smoothly as possible to guarantee stability and save energy, and motion efficiency, which means that the robot is required to pass through the required states as soon as possible to give way to others. Second, the robot's kinematics and mechanical constraints are given in (4b), (4c), and (4d), where  $v_{\max}$ ,  $a_{\min} (< 0)$ , and  $a_{\max} (> 0)$  are the maximal speed, maximal deceleration, and maximal acceleration of the robot, respectively. At last, the constraint for collision and deadlock avoidance is given in (4e). Indeed, (4e) means that the actual motion time at the current state should be larger than the predicted minimal motion time  $t_w(i)$ .

$$\min_{a^i} w_1 \sqrt{\int_{l_c}^{l_s^i} a^i(l)^2 dl} + w_2 \int_{l_c}^{l_s^i} \frac{1}{v^i(l)} dl \quad (4a)$$

subject to:  $\forall L \in [l_c, l_s^i]$ ,

$$\frac{v^i(L)^2}{2} = \frac{v^i(l_c)^2}{2} + \int_{l_c}^L a^i(l) dl, \quad (4b)$$

$$0 \leq v^i(L) \leq v_{\max}, \quad (4c)$$

$$a_{\min} \leq a^i(L) \leq a_{\max}, \quad (4d)$$

$$t_w(i) \leq \int_{l_c}^{l_s^i} \frac{1}{v^i(l)} dl. \quad (4e)$$

Since we focus on path-horizon MPC, the path segment  $P_s^i$  can be discretized into a set of equal-length sub-segments:  $L_0, L_1, \dots, L_K$ , where  $L_0 = 0$ ,  $L_K = l_s^i$ ,  $h = l_s^i/K$ , and  $\forall k \in \mathbb{K} = \{0, 1, \dots, K\}$ ,  $L_k = kh$ . The current discrete instant is denoted as  $k_c$ , satisfying  $l_c = k_c h$ . Based on the path discretization, the control variable  $a^i(L)$  is discretized via piece-wise constant:  $\forall L \in [L_k, L_{k+1}]$ ,  $a^i(L) = a^i(L_k)$ . To guarantee safety, the path step length  $h$  should satisfy that in any situation, the robot can stop its motion before reaching the next discrete point. Note that in the worst case, the minimal motion distance that the robot needs to stop itself is:  $v_{\max}^2/|2a_{\min}|$ . Hence,  $h$  should satisfy:  $h \geq v_{\max}^2/|2a_{\min}|$ .

Let  $b^i(L) = v^i(L)^2$ ,  $b_k^i = b^i(L_k)$ , and  $a_k^i = a^i(L_k)$ ,  $\forall k \in \mathbb{K}$ . Substituting them into (4b), we have

$$b^i(L) = b_k^i + 2a_k^i(L - L_k), \forall L \in (L_k, L_{k+1}]. \quad (5)$$

Hence,

$$\int_{l_c}^{l_s^i} \frac{1}{v^i(l)} dl = \sum_{k=k_c}^{K-1} \int_{L_k}^{L_{k+1}} \frac{1}{v^i(l)} dl = \sum_{k=k_c}^{K-1} \frac{2h}{\sqrt{b_k^i} + \sqrt{b_{k+1}^i}}$$

Hence, we can reformulate (4) using a discrete form, namely (6) in terms of (6a)–(6e).

$$\min_{\mathbf{a}^i, \mathbf{b}^i} w_1 \sqrt{h} \|\mathbf{a}^i\|_2 + w_2 \sum_{k=k_c}^{K-1} \frac{2h}{\sqrt{b_{k+1}^i} + \sqrt{b_k^i}} \quad (6a)$$

$$\text{subject to: } \mathbf{A}\mathbf{b}^i - 2h\mathbf{a}^i = 0, \quad (6b)$$

$$b_{k_c}^i = v^i(k_c)^2, \quad \mathbf{0} \leq \mathbf{b}^i \leq v_{\max}^2 \mathbf{1}, \quad (6c)$$

$$a_{\min} \mathbf{1} \leq \mathbf{a}^i \leq a_{\max} \mathbf{1}, \quad (6d)$$

$$t_w(i) - \sum_{k=k_c}^{K-1} \frac{2h}{\sqrt{b_{k+1}^i} + \sqrt{b_k^i}} \leq 0. \quad (6e)$$

where  $\mathbf{b}^i = (b_{k_c}^i, b_{k_c+1}^i, \dots, b_K^i)^T$  and  $\mathbf{a}^i = (a_{k_c}^i, a_{k_c+1}^i, \dots, a_{K-1}^i)^T$  are control variables,  $\mathbf{A} = (A_{kj})_{(K-k_c) \times (K-k_c+1)}$  satisfies  $A_{kk} = -1$  and  $A_{k,k+1} = 1$  for  $k = 1, \dots, K - k_c$ , while others are 0,  $v^i(k_c)$  is the speed at the current discrete instant  $k_c$ , and  $\mathbf{0}$  and  $\mathbf{1}$  are the all-zero and all-one vectors with proper dimensions, respectively. Moreover, since there may exist a situation that the robot should stop to wait for others, which means that  $\exists k \in \{k_c, k_c + 1, \dots, K - 1\}$  such that  $v_k^i = v_{k+1}^i = 0$ , resulting in  $\frac{1}{\sqrt{b_{k+1}^i} + \sqrt{b_k^i}}$  is infinite. To deal with this situation, we set a large value, e.g.,  $10^6$ , to approximately represent 1/0.

In (4), the objective and constraints satisfy a convex problem except for (6e), which is a difference between two convex functions. Thus, the problem can be solved approximately via sequential convex programming (SCP) [38] or incremental SCP [23]. Since there is only one non-convex constraint, i.e., (6e), the two methods are with the same efficiency in practice. Hence, the SCP approach is adopted to resolve our problem. In the sequel, we describe the convex approximation of (6) at a given point and give the detailed procedure to solve (6).

Let  $g(\mathbf{b}^i) = \sum_{k=k_c}^{K-1} \frac{2h}{\sqrt{b_{k+1}^i} + \sqrt{b_k^i}}$ , and  $\nabla g(\mathbf{b}^i)$  is the gradient of  $g(\mathbf{b}^i)$ . Its first-order Taylor expansion at a given point  $\mathbf{b}_m^i$  can be described as  $g(\mathbf{b}_m^i) + \nabla g(\mathbf{b}_m^i)^T (\mathbf{b}^i - \mathbf{b}_m^i)$ . Hence, the approximate convex problem at  $\mathbf{b}_m^i$ , denoted as  $P(\mathbf{b}_m^i)$ , can be described as follows.

$$\begin{aligned} & \min_{\mathbf{a}^i, \mathbf{b}^i} w_1 \sqrt{h} \|\mathbf{a}^i\|_2 + w_2 \sum_{k=k_c}^{K-1} \frac{2h}{\sqrt{b_{k+1}^i} + \sqrt{b_k^i}} \\ & \text{subject to: } \mathbf{A}\mathbf{b}^i - 2h\mathbf{a}^i = 0, \quad b_{k_c}^i = v^i(k_c)^2, \quad P(\mathbf{b}_m^i) \\ & \quad \mathbf{0} \leq \mathbf{b}^i \leq v_{\max}^2 \mathbf{1}, \quad a_{\min} \mathbf{1} \leq \mathbf{a}^i \leq a_{\max} \mathbf{1}, \\ & \quad t_w(i) - [g(\mathbf{b}_m^i) + \nabla g(\mathbf{b}_m^i)^T (\mathbf{b}^i - \mathbf{b}_m^i)] \leq 0. \end{aligned}$$

Hence, (6) can be resolved iteratively by resolving  $P(\mathbf{b}_m^i)$ . Algorithm 4 gives the detailed SCP procedure. Lines 3–12 are the iteration process of SCP, while Lines 13–15 focus

---

**Algorithm 4:** SCP procedure to solve (6).

---

**Input :** Current speed  $v^i(k_c)$ , minimal motion time  $t_w(i)$ , number of steps  $K$ , step length  $h$ , maximal number of iterations  $M$ , and precision  $\epsilon$ .

**Output:**  $\mathbf{b}^i$  and  $\mathbf{a}^i$ .

```

1 Initialization:  $m = 0$ ,  $\mathbf{b}_m^i = \mathbf{0}$ ,  $F_m^i = 0$ ;
2 while  $m \leq M$  do
3   Compute  $g(\mathbf{b}_m^i)$  and  $\nabla g(\mathbf{b}_m^i)$  ;
4   Construct approximate convex problem  $P(\mathbf{b}_m^i)$ ;
5   Solve  $P(\mathbf{b}_m^i)$ ;
6   if  $P(\mathbf{b}_m^i)$  is well solved then
7     Retrieve the optimal solution  $\mathbf{b}^i$  and  $\mathbf{a}^i$ , and the
      optimal value  $F^i$ ;
8     if  $\|\mathbf{b}^i - \mathbf{b}_m^i\|_2 \leq \epsilon$  or  $|F^i - F_m^i| \leq \epsilon$  then
9       return  $\mathbf{b}^i$  and  $\mathbf{a}^i$ ;
10    else
11       $m = m + 1$ ;
12       $F_m^i = F^i$ ;  $\mathbf{b}_m^i = \mathbf{b}^i$ ;
13    else
14       $\forall k \in \{k_c, \dots, K - 1\}$ ,
         $a_k^i = -v^i(k_c)^2 / (2 * (K - k_c) * h)$ ,
         $b_{k+1}^i = b_k^i - v^i(k_c)^2 / (K - k_c)$ ;
15  return  $\mathbf{b}^i$  and  $\mathbf{a}^i$ ;

```

---

on the situation that  $P(\mathbf{b}_m^i)$  cannot find an optimal solution at an iteration. For each iteration, the value of  $\mathbf{b}_m^i$  is set as the optimal solution of the former iteration (Line 12). We give two stopping criteria of the iteration process: The number of iterations reaches the maximal one (Line 2), and the difference between two successive solutions is less than the given precision (Line 8). The iteration is stopped if either of them is satisfied. If  $P(\mathbf{b}_m^i)$  cannot find an optimal solution at an iteration, we generate a feasible solution to stop the robot before it reaches the end of the current state (Lines 13–15).

Finally, Algorithm 5 shows the complete hybrid MPC-based motion control at an arbitrary state based on Algorithms 2, 3, and 4. Line 3 performs the discrete decision-making to determine whether the robot can enable or fire its current transition. Lines 4–6 compute the minimal motion time if the robot cannot enable its current transition due to collision and deadlock avoidance, while Lines 7 and 8 compute the minimal motion time if the robot can enable its current transition but does not win the right to fire the transition during negotiation. Once the minimal motion time is obtained, the robot computes the optimal motion at the current step (Line 11). Note that the computation returns a sequence of motion commands in the future, but the robot applies only the first one to move to the next discrete point (Lines 12–14). Once it arrives at the next point, the robot updates its status (Lines 15 and 16) and starts a new iteration.

It is worth noting that the computational complexity of Algorithms 2 and 3 is  $O(n)$ , while Algorithm 4 includes the computation of a finite number of convex optimization problems, each of which can be resolved approximately in polynomial time with respect to the number of variables and the size of the optimization problem [23], [39]. Hence, at each instant, Algorithm 5 can be resolved with a polynomial-time complexity with respect to the numbers of robots and prediction steps. Moreover, according to the scalability metric

**Algorithm 5:** MPC-based control for  $r_i$ 's motion at  $s$ .

---

```

Input : Mechanical constraints:  $v_{\max}$ ,  $a_{\max}$ , and  $a_{\min}$ ,  

        hybrid state at the tail of  $P_s^i$ :  $(s, p_0^i, v_0^i, 0)$ , and the  

        precision:  $\epsilon$ .
1 Initialization: Path discretization by setting  $h$  and  $K$ ,  $k_c = 0$ ;
2 while  $k_c < K$  do
3    $\{\mathcal{R}_Y, \mathcal{R}_N, \Gamma, \gamma\} = \text{Algorithm 2};$ 
4   if  $\gamma = 0$  then
5     Determine enable-dependent robots  $\mathcal{R}^i(s);$ 
6     Compute the minimal motion time  $t_w(i)$  via (3);
7   else if  $\gamma = 1$  then
8      $t_w(i) = \text{Algorithm 3};$ 
9   else if  $\gamma = 2$  then
10     $t_w(i) = 0;$ 
11 Call Algorithm 4 and return  $\mathbf{b}^i, \mathbf{a}^i;$ 
12  $a_{k_c}^i = \mathbf{a}^i(1); /* \text{select the first value. */}$ 
13  $t[k_c \rightarrow k_c + 1] = \frac{\sqrt{\mathbf{b}^i(2)} - \sqrt{\mathbf{b}^i(1)}}{a^i[k_c]};$ 
14 Move to  $L_{k_c+1}$  with acceleration  $a_{k_c}^i$  and speed  $v_{k_c}^i;$ 
15  $k_c = k_c + 1$  and  $v_{k_c}^i = \sqrt{\mathbf{b}^i(2)}$ ;
16 Update the hybrid state  $(s, p_{k_c}^i, v_{k_c}^i, L - k_c h).$ 

```

---

defined in [40], the system is scalable under Algorithm 5. In our situation, given a system with  $N$  robots, the value rate produced by the system is defined as the rate of motion task finished per time unit, computed as  $\mathbb{V}(N) = Ne_0$ , where  $e_0$  is the motion efficiency of each robot, and the cost rate of the system is defined as the maximal communication time of a robot to receive the responses from its neighbors to each request, computed as  $\mathbb{C}(N) = (N - 1)T_{com}$ , where  $T_{com}$  is the maximal communication time between any pair of robots. As there are at least two robots in an MMRS, the scalability metric  $\varphi(N)$  can be described as  $\varphi(2, N)$ . Hence, we have

$$\begin{aligned} \varphi(N) &= \varphi(2, N) = \frac{\mathbb{V}(N)/\mathbb{C}(N)}{\mathbb{V}(2)/\mathbb{C}(2)} \\ &= \frac{Ne_0/((N - 1)T_{com})}{2e_0/T_{com}} = \frac{1}{2} + \frac{1}{2(N - 1)}, \end{aligned} \quad (7)$$

and  $\lim_{N \rightarrow \infty} \varphi(N) = \frac{1}{2}$ . Therefore, the system is scalable.

### C. Effectiveness Analysis of the Hybrid Method

This subsection analyzes the effectiveness of the proposed hybrid method. Given a robot  $r_i$ , let  $t_p(i)$  and  $\bar{t}^i$  be the predicted and actual motion time that  $r_i$  needs to move at the current state. Moreover, let  $F_1^i = w_1 \sqrt{h} \|\mathbf{a}^i\|_2$  and  $F_2^i = w_2 \sum_{k=k_c}^{K-1} \frac{2h}{\sqrt{b_{k+1}^i} + \sqrt{b_k^i}}$ , then objective function (6a) can be written as  $F^i = F_1^i + F_2^i$ .

**Lemma 1.** If  $r_i$  can fire its current transition, then  $t_p(i) \geq \bar{t}^i$ .

*Proof:* If  $r_i$  can fire its current transition, constraint (6e) does not need to be considered. In this situation, decelerated motion is not an optimal solution. Indeed, let  $F^i$  and  $\hat{F}^i$  be the objective values under constant and decelerated motion with respect to the current speed, respectively. First, it is clear that  $F_1^i = 0$ , while  $\hat{F}_1^i > 0$ . Second, the motion time to the same position under a decelerated motion is larger than that under a constant motion, which means  $F_2^i < \hat{F}_2^i$ . Thus,  $F^i < \hat{F}^i$ . It means that in this situation,  $r_i$  should move with its current speed or an accelerated speed. Hence,  $t_p(i) \geq \bar{t}^i$ . ■

Based on Lemma 1, we can derive the following lemma.

**Lemma 2.** If a robot can move at its current speed  $v_c$ , its optimal motion is either a constant or an accelerated motion with respect to  $v_c$ .

**Lemma 3.** The motion under the solution of problem (6) can guarantee the avoidance of robot collisions and system deadlocks.

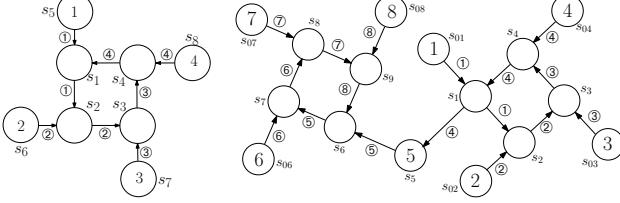
*Proof:* Suppose that  $r_{i_0}$  cannot transit to the next state at its current state  $s$ , and the sequence of its enable-dependent robots is  $\mathcal{R}^i(s) = \{r_{i_1}, r_{i_2}, \dots, r_{i_j}\}$ , where  $r_{i_k}$  needs to wait for the move of  $r_{i_{k+1}}$  for  $k = 1, 2, \dots, j - 1$ , and  $r_{i_j}$  can fire its current transition.  $r_{i_k}$ 's real motion time to the required position is denoted as  $\bar{t}^{i_k}$ . Based on Lemma 1, we have  $t_p(i_j) \geq \bar{t}^{i_j}$ . If  $t_p(i_{j-1}) \geq t_p(i_j)$ ,  $r_{i_{j-1}}$  can move at least with its current speed based on Lemma 2. Moreover, based on constraint (6e),  $\bar{t}^{i_{j-1}} \geq t_p(i_j)$ . Thus, we have  $t_p(i_{j-1}) \geq \bar{t}^{i_{j-1}} \geq t_p(i_j) \geq \bar{t}^{i_j}$ ; If  $t_p(i_{j-1}) < t_p(i_j)$ ,  $r_{i_{j-1}}$  needs to decelerate its motion based on its own optimal problem (6). In this case,  $r_{i_{j-1}}$ 's optimal solution should reach the boundary of its own constraint (6e), implying  $\bar{t}^{i_{j-1}} = t_p(i_j) > \bar{t}^{i_j}$ . Hence, the real motion time of  $r_{i_{j-1}}$  satisfies  $\max\{t_p(i_{j-1}), t_p(i_j)\} \geq \bar{t}^{i_{j-1}} \geq \bar{t}^{i_j}$ .

Suppose that  $r_{i_{k+1}}$  satisfies  $\max\{t_p(i_{k+1}), \dots, t_p(i_j)\} \geq \bar{t}^{i_{k+1}} \geq \dots \geq \bar{t}^{i_j}$ . Let us consider  $r_{i_k}$ . If  $t_p(i_k) \geq \max\{t_p(i_{k+1}), \dots, t_p(i_j)\}$ ,  $r_{i_k}$  at least can move at its current speed based on its own (6). Thus, we have  $t_p(i_k) \geq \bar{t}^{i_k} \geq \max\{t_p(i_{k+1}), \dots, t_p(i_j)\} \geq \bar{t}^{i_{k+1}}$ . Note that in this case  $t_p(i_k) = \max\{t_p(i_k), t_p(i_{k+1}), \dots, t_p(i_j)\}$ . If  $t_p(i_k) < \max\{t_p(i_{k+1}), \dots, t_p(i_j)\}$ ,  $r_{i_k}$  needs to decelerate its motion. In this case,  $r_{i_k}$ 's optimal solution should reach the boundary of its own constraint (6e). This means  $\bar{t}^{i_{k-1}} = \max\{t_p(i_{k+1}), \dots, t_p(i_j)\} > \bar{t}^{i_{k+1}}$ . In conclusion, we have  $\max\{t_p(i_k), \dots, t_p(i_j)\} \geq \bar{t}^{i_k} \geq \dots \geq \bar{t}^{i_{k+1}} \geq \dots \geq \bar{t}^{i_j}$ .

Based on the method of induction, we have that  $\bar{t}^{i_0} \geq \bar{t}^{i_k} \geq \dots \geq \bar{t}^{i_{k+1}} \geq \dots \geq \bar{t}^{i_j}$ . It means that when  $r_{i_0}$  arrives at the end of its current state, its enable-dependent robots have left their required positions. Hence,  $r_{i_0}$  can transit to its next state without any collision or deadlock. ■

**Lemma 4.** Algorithm 4 can always return a sub-optimal, if not optimal, solution to (6).

*Proof:* Based on our discretization, each robot can stop its motion from  $L_{k_c}$  to  $L_K$ . Thus, when  $P(\mathbf{b}_m^i)$  fails to find an optimal solution, Lines 13 - 15 in Algorithm 4 can guarantee a feasible solution of (6). Otherwise, the optimal solution of  $P(\mathbf{b}_m^i)$  at  $\mathbf{b}_m^i$  is a sub-optimal, if not optimal, solution of (6). Indeed, for a convex function  $f(\mathbf{x})$ , given an arbitrary point  $\mathbf{x}_0$ , we have  $\forall \mathbf{x}, f(\mathbf{x}) \geq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0)$ . Since  $g(\mathbf{b}^i) = \sum_{k=k_c}^{K-1} \frac{2h}{\sqrt{b_{k+1}^i} + \sqrt{b_k^i}}$  is a convex function, we have  $g(\mathbf{b}^i) \geq g(\mathbf{b}_m^i) + \nabla g(\mathbf{b}_m^i)^T(\mathbf{b}^i - \mathbf{b}_m^i)$ . Hence,  $t_w(i) - g(\mathbf{b}^i) \leq t_w(i) - [g(\mathbf{b}_m^i) + \nabla g(\mathbf{b}_m^i)^T(\mathbf{b}^i - \mathbf{b}_m^i)]$ . Compared with  $P(\mathbf{b}_m^i)$  and (6), the optimal solution of  $P(\mathbf{b}_m^i)$  is a feasible solution of (6). The sub-optimality can be derived directly from the local convergence of SCP [41]. ■



(a) Transition system. (b) A more sophisticated transition system.

Fig. 4. The transition systems studied in the experiments.

**Theorem 1.** Under Algorithm 5, each robot can move under a sub-optimal, if not optimal, motion without causing collisions and deadlocks.

*Proof:* It is easily proved based on Lemmas 3 and 4 since Lemma 3 guarantees collision and deadlock avoidance at each time instant of the MPC procedure and Lemma 4 guarantees a sub-optimal, if not optimal, motion. ■

## V. SIMULATION STUDY

In the sequel, we show some simulation results and a real-world experiment for MMRSSs under the control of the proposed hybrid method. Our simulations are implemented by MATLAB with the CVX toolbox and the MOSEK solver on the HP Z440 workstation, whose CPU and memory are Intel(R) Xeon(R) E5-1650 v3 @ 3.50GHz and 16.0 GB, respectively. Our real-world experiment is done with three TurtleBot3 Waffle Pi robots.

### A. Simulation Results with Our Hybrid Control Method

Our first system for simulation is given in Fig. 1(a), whose transition system is shown in Fig. 4(a). The private states  $s_5$ ,  $s_6$ ,  $s_7$ , and  $s_8$  represent the path segments from the vehicles' current locations to the intersection boundaries  $a$ ,  $g$ ,  $d$ , and  $j$ , respectively, and they have the same path length  $3\rho$ , where  $\rho = 100$  distance units.  $r_1$ 's collision path segments  $ab$  and  $bc$  are modeled as the collision states  $s_1$  and  $s_2$ , respectively.  $r_2$ 's collision path segments  $gh$  and  $hi$  are represented by the collision states  $s_2$  and  $s_3$ , respectively. Similarly,  $de$  and  $ef$  are modeled as  $s_3$  and  $s_4$  for  $r_3$ , and  $jk$  and  $km$  are modeled as  $s_4$  and  $s_1$  for  $r_4$ . We assume that all collision path segments have the same path length  $4\rho$ . Moreover, the mechanical constraints of the four vehicles are  $v_{\max} = 100$  distance units/sec,  $a_{\min} = -150$  distance units/sec<sup>2</sup>, and  $a_{\max} = 150$  distance units/sec<sup>2</sup>. The discretization step is  $h = 100/3$  distance units, generating 9 and 12 discrete steps at a private state and a collision state, respectively. During our simulation, the computation time of each iteration in Algorithm 4 is about 0.44s, and most of the computation can be convergent in 2 iterations. However, the number of iterations generated at some instants is larger than 10. Note that the number of iterations highly depends on the initial solution. It will be our future work to improve the quality of the initial solution.

In the simulation, the initial states of  $r_1$ ,  $r_2$ ,  $r_3$ , and  $r_4$  are  $s_5$ ,  $s_6$ ,  $s_7$ , and  $s_8$ , respectively, and their initial speeds are 60, 50, 40, and 30 distance units/sec, respectively. Fig. 5 shows the speed profile of the four robots during their motion,

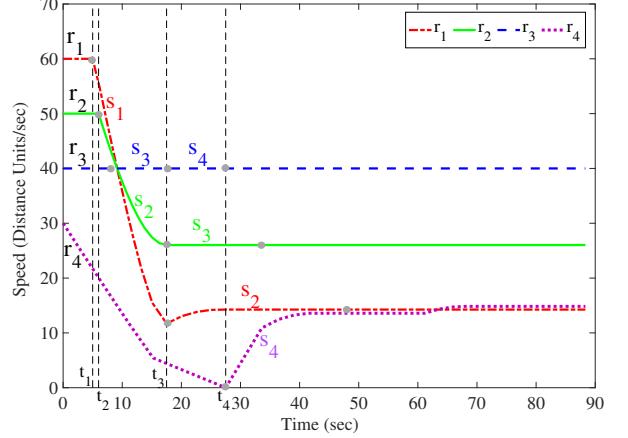


Fig. 5. Speed of the four robots in the simulation.

whether each grey dot denotes that there is a state transition at the corresponding time instant. First, since  $s_1$ ,  $s_2$ ,  $s_3$ , and  $s_4$  form a collision region  $X$ , the four vehicles need negotiation according to Algorithm 1. Based on the negotiation process, since  $r_1$ ,  $r_2$ , and  $r_3$  have higher speed than  $r_4$ , they can fire their current transitions and move into  $X$  first, while  $r_4$  should wait for the move of  $r_3$ . Hence, as shown in Fig. 5, at the start,  $r_1$ ,  $r_2$ , and  $r_3$  continue their motion at their current speeds, while  $r_4$  slows down.

During the system's evolution,  $r_1$  first transits to  $s_1$  at time  $t_1$ . Since  $r_2$  will arrive at  $s_2$  earlier than  $r_1$ , the negotiation among  $r_1$  -  $r_4$  decides that  $r_1$  should wait for  $r_2$ 's move from  $s_2$ . Note that  $r_4$  is still waiting for  $r_3$ . Hence,  $r_1$  slows down its motion at  $t_1$ , as shown in Fig. 5. Subsequently,  $r_2$  transits to  $s_2$  at  $t_2$ . Since  $r_3$  will transit to  $s_3$  earlier than  $r_2$ ,  $r_2$  needs to wait for  $r_3$  at  $s_2$ . Hence,  $r_2$  also starts to decrease its speed at  $t_2$ . At time instant  $t_3$ ,  $r_1$ ,  $r_2$ , and  $r_3$  arrive at the end of  $s_1$ ,  $s_2$ , and  $s_3$  simultaneously. Thus, when  $r_3$  transits to  $s_4$ ,  $r_2$  transits to  $s_3$  and  $r_1$  to  $s_2$ . Based on their optimal objectives,  $r_1$  first accelerates and then keeps a constant speed, while  $r_2$  moves with its current speed. Note that  $r_4$  still decelerates to avoid collision with  $r_3$ . At time instant  $t_4$ ,  $r_3$  arrives at the end of  $s_4$  and then moves away. So  $r_4$  transits to  $s_4$ . Since it does not need to wait for any robot anymore,  $r_4$  first speeds up and then keeps a uniform motion. During the whole simulation, the minimal distance between any two robots is around 5 distance units (between  $r_3$  and  $r_4$ ). The simulation video can be found at <https://yuanzhou-yzhou.github.io/hybrid-motion/>.

In conclusion, from the speed profiles given in Fig. 5, we can find that during the motion in the intersection, each robot can avoid collisions and deadlocks by adjusting its speed while keeping its motion as smooth as possible.

### B. Comparison with Discrete Control

To illustrate the efficiency of our method, we compare the robot motion computed by our method and the discrete one. For the discrete method, we introduce a straightforward way to control robot motion at a discrete state: The robot first moves at a constant speed; when the robot checks that it cannot move to the next state, it slows down its motion at the largest deceleration and stops at the end of the state. Moreover, the

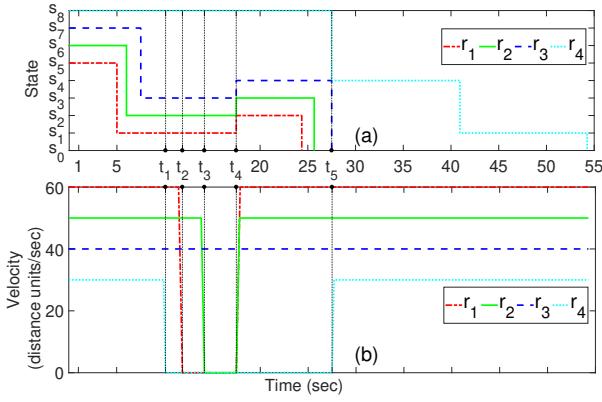


Fig. 6. Simulation results under only discrete control.

robots transit to a state on a first-arrive first-transit basis as long as its transition does not cause collisions or deadlocks. In this way, we can simulate the motion of the robots in Fig. 4(a). The results are given in Fig. 6, where (a) and (b) show the state transitions and speed profiles, respectively.

From Fig. 6(a), we can find that \$r\_1, r\_2\$, and \$r\_3\$ sequentially move into the intersection and arrive at \$s\_1, s\_2\$, and \$s\_3\$, respectively, while \$r\_4\$ is still at \$s\_8\$. When \$r\_4\$ reaches the end of \$s\_8\$, its discrete controller checks a deadlock and thus stops the robot's motion immediately at the largest deceleration. Hence, as given in Fig. 6(b), \$r\_4\$ stops its motion at the end of \$s\_8\$ at time \$t\_1\$, while \$r\_1, r\_2\$, and \$r\_3\$ are still moving at \$s\_1, s\_2\$, and \$s\_3\$, respectively. Near the end of \$s\_1\$, \$r\_1\$'s discrete controller predicts that \$r\_2\$ is still at \$s\_2\$ when \$r\_1\$ reaches the end of \$s\_1\$. Thus, \$r\_1\$ slows down its motion and stops at the end of \$s\_1\$ at time \$t\_2\$, as given in Fig. 6(b). Similarly, \$r\_2\$'s controller predicts that when \$r\_2\$ arrives at the end of \$s\_2\$, \$r\_3\$ is still at \$s\_3\$, so \$r\_2\$ slows down its motion with its maximal deceleration and completely stops at time \$t\_3\$. \$r\_3\$ continues its motion at \$s\_3\$ until \$t\_4\$, at which the robot transits to \$s\_4\$. After \$r\_3\$ moves to \$s\_4\$, \$r\_2\$ can move to \$s\_3\$, and then \$r\_1\$ can move to \$s\_2\$, but \$r\_4\$ is still stopping at \$s\_8\$. Hence, \$r\_1\$ and \$r\_2\$ resume their motion to their former speeds with the maximal acceleration and then move at constant speeds. At time \$t\_5\$, \$r\_3\$ moves away from \$s\_4\$, so \$r\_4\$ can move forward. Speeding up with its maximal acceleration, \$r\_4\$ resumes its motion and then moves at a constant speed.

From Figs. 5 and 6(b), we can find that the motion generated by the proposed hybrid method has much fewer stops and jerks than that generated by discrete control. Hence, the proposed method can generate smooth motion for robots. When a robot detects that a collision or deadlock occurs if it were at the next state, the robot should wait for a proper duration at its current state such that other robots can give way to it. In discrete control, the robot adjusts its speed only when it is near the end of the state. It means the robot has a shorter time and path length to adjust its speed, so it has to slow down at its maximal deceleration and stop at the end of the current state. In our method, the robot first predicts whether it can transit to the next state at any discrete point and then optimizes its speed based on the predicted minimal motion time. In this way, the robot can smoothly arrive at the end of the state with an intelligently-tuned smooth speed change. Hence, our approach leads to the advantages of fewer stops and jerks.

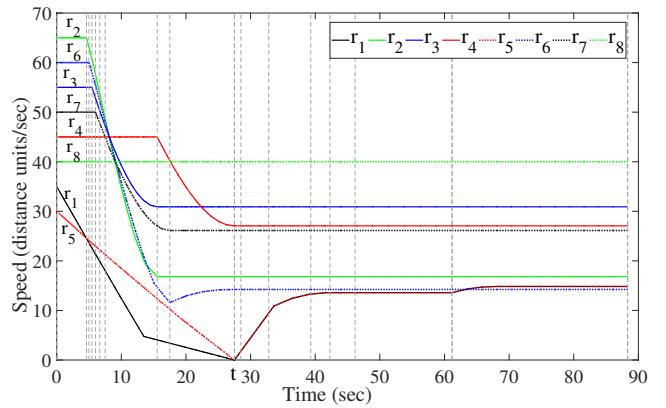


Fig. 7. Speed evolution of the robots.

### C. A More Complex Scenario

This subsection studies a more complex system to show the efficiency of the proposed hybrid method. As given in Fig. 4(b), there are 8 robots, \$r\_1\$–\$r\_8\$, in the system, and they are currently at their private states \$s\_{01}\$–\$s\_{08}\$. The system parameters in this scenario are the same as the system described in Section V-A. The initial speeds of \$r\_1\$–\$r\_8\$ are 35, 65, 55, 45, 30, 60, 50, and 40 distance units/sec, respectively. Clearly, the collision region in this scenario is \$X = \{s\_1, s\_2, \dots, s\_9\}\$.

Fig. 7 shows the speed profiles of these robots. The vertical lines indicate the time instants when the robots fire their transitions. First, based on the negotiation process, \$r\_1\$ and \$r\_6\$ need to wait for \$r\_4\$ and \$r\_8\$, respectively. Hence, the hybrid controllers decide that \$r\_1\$ and \$r\_6\$ should slow down, as shown in Fig. 7. Second, \$r\_2\$ first transits to \$s\_2\$ at the time denoted by the first vertical line. When it arrives at \$s\_2\$, \$r\_2\$ should slow down since the negotiation process finds that \$r\_3\$ moves to \$s\_3\$ earlier. Next, \$r\_6\$ transits to \$s\_7\$ and then reduces its speed since \$r\_7\$ will reach \$s\_8\$ earlier. Sequentially, \$r\_3, r\_7\$, and \$r\_4\$ transit to their next states and decrease their speeds to avoid collisions. Similarly, we can analyze the motion of these robots at different time instants. When \$r\_8\$ arrives at its private state at time \$t\$, \$r\_5\$ can move to \$s\_6\$, and \$r\_4\$ to \$s\_5\$, resulting in \$r\_1\$'s transiting to \$s\_1\$. Hence, the states of these robots at time \$t\$ are \$s\_1, s\_3, s\_4, s\_5, s\_6, s\_8, s\_9\$, and the private state \$pvt\$, respectively. From \$t\$, all robots can move forward directly, so \$r\_1\$ and \$r\_5\$ need to resume their motion. Since their parameters are the same, their optimal speeds are the same. It can be seen in Fig. 7.

### D. Real-World Implementation

In this section, we evaluate our algorithm with three physical TurtleBot3 Waffle Pi robots. Each robot is equipped with a Raspberry Pi computer with Ubuntu 16.04 and ROS Kinetic, a Raspberry Pi Camera Module 2 and a 360 Laser Distance Sensor LDS-01. Each robot is equipped with a remote computer to execute the control algorithms and generate motion commands. The physical paths of the three robots are shown in Fig. 8. The initial speeds of the three robots are 5.5 cm/sec, 9.5 cm/sec, and 17.5 cm/sec, respectively. We first conduct an experiment where each robot is controlled with only the collision avoidance algorithm and then conduct the second experiment where each robot is controlled

TABLE I  
COMPARISONS OF THE MOST RELEVANT METHODS

Metric	Method		
	$M_1$ ([13])	$M_2$ ([14])	$M_I$ (Ours)
Addressed Deadlock	Decision Deadlocks	Decision Deadlocks	Physical Deadlocks
Deadlock Resolution	Compute motion priorities based on different motion time	Apply the predefined priorities	Adjust each robot's speed individually
Motion Manner	Distributed Computation, Synchronous Motion	Distributed Computation, Synchronous Motion	Distributed Computation, Asynchronous Motion
Computation Cost		$\mathcal{C}(M_1) \leq \mathcal{C}(M_2) \leq \mathcal{C}(M_I)$	
Permissive Motion		$\mathcal{S}(M_I) \geq \mathcal{S}(M_2) \geq \mathcal{S}(M_1)$	

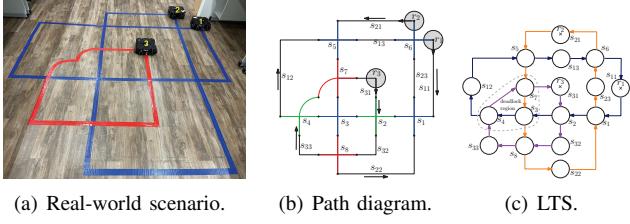


Fig. 8. The physical experimental scenario.

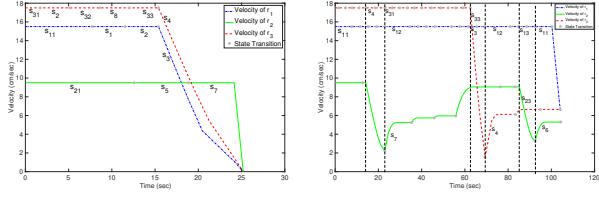


Fig. 9. The speed profiles of the robots under different controllers.

by our proposed controller. Fig. 9 gives the speed profiles of the three robots under the two experiments. The results show that the three robots are in a deadlock in the first experiment, while they can move persistently in the second experiment. The detailed experimental results can be found at <https://yuanzhou-yzhou.github.io/hybrid-motion/>.

## VI. DISCUSSION AND COMPARISON

In this section, we compare our method (denoted as  $M_I$ ) with the two most relevant methods: [13] and [14], denoted as  $M_1$  and  $M_2$ , respectively. We compare these methods from the perspectives of deadlock to be addressed, resolution methods, computation cost, and permissive motions. The comparison results are shown in Table I, where  $\mathcal{C}()$  and  $\mathcal{S}()$  denote the computation cost and permissive motions, respectively. The main difference between their works and ours is that they focus on decision deadlocks, while we aim to deal with physical deadlocks, which are more serious and difficult to resolve.

Indeed, there is a trade-off between the computation cost and permissive motion.  $M_1$  abstracts intersecting path segments into a set of nonadjacent collision zones. It means that there is at least one collision-free position for any robot between any two collision zones. Hence, no physical deadlocks will occur during collision avoidance. This method is efficient but may forbid many permissive motions.  $M_2$  studies conflict resolution in road junctions. Instead of partitioning the paths directly, it divides an intersection into a set of collision zones and then computes the time duration to pass the collision zones

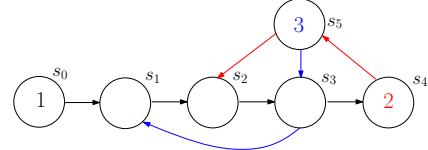


Fig. 10. An example for the comparison of different methods, where  $r_1 : s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4$ ,  $r_2 : s_4 \rightarrow s_5 \rightarrow s_2$ , and  $r_3 : s_5 \rightarrow s_3 \rightarrow s_1$ .

for each vehicle along its path. Based on the time duration,  $M_2$  can determine the temporal advantages of the vehicles, which determine the orders of vehicles to pass through collision zones. Similar to  $M_1$ , there may exist cycles in the temporal advantages, resulting in decision deadlocks. They can be resolved by the predefined vehicle priorities. It is worth noting that decision deadlocks may not result in physical deadlocks (an example will be given later). Hence, some feasible motions are also forbidden. In our method, we study physical deadlock avoidance directly. It means that only when its motion to the next state causes a physical deadlock does a robot need to change its speed profile.

We show an example to demonstrate the difference in permissive motions of the three methods. As shown in Fig. 10, there are 3 robots passing through an intersection divided into 5 collision zones:  $s_1-s_5$ . Based on the method in [13], the zones are abstracted as a single collision zone. At any time instant, only one robot can be at one of these zones. Suppose that the current time durations of robots at their states are  $T_1^1 = [0, 2]$ ,  $T_2^1 = [2, 4]$ ,  $T_3^1 = [4, 6]$ ,  $T_1^2 = [6, 8]$ ,  $T_2^2 = [8, 10]$ ;  $T_3^2 = [0, 2]$ ,  $T_1^3 = [2, 5]$ ,  $T_2^3 = [5, 8]$ , where  $T_j^i$  means the time duration for  $r_i$  to pass through  $s_j$ . Based on the method in [14], the original temporal advantages are  $r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow r_1$ . After deadlock resolution, the sequence of yieldings is  $r_1$  yields  $r_2$  and  $r_3$ , and  $r_2$  yields  $r_3$ . Hence,  $r_1$  cannot move to  $s_1$  until  $r_3$  leaves  $s_1$ . However, in our method, the original temporal advantages are allowable since there are no physical deadlocks. Based on our method, after 2 time units,  $r_1$  enters  $s_1$ , and  $r_3$  enters  $s_3$ , while after 3 time units,  $r_2$  moves to  $s_5$ . Before  $r_3$  enters  $s_1$ ,  $r_1$  has left  $s_1$  and entered  $s_2$ . When  $r_1$  leaves  $s_2$  after 6 time units,  $r_2$  is ready to move to  $s_2$ . Hence, all the robots can move at their current speed profiles.

## VII. CONCLUSION

This paper investigates motion control of MMRSs where each robot is required to move along a fixed path and proposes a distributed, real-time, and hybrid method. First, the motion of each robot is modeled as a discrete transition system,

and an online distributed policy is designed to determine whether a robot can fire its current transition. Third, an MPC-based policy is proposed to optimize the speed of each robot at each discrete state to obey the discrete decision. Each optimization problem constructed at this phase only contains the mechanical constraints of the robot and the time-related constraint generated from the discrete control phase. In this way, we can reduce the scale of the local optimization problem significantly at any step.

In the future, we will investigate the implementation of our method on real robots. Our approach will be extended to systems where a robot has multiple paths or systems operating in a free environment with complex tasks (e.g., high-dynamics formation [42]) or mechanisms (e.g., soft robots [43]). Another interesting topic is to investigate systems where a path has multiple robots. Moreover, optimization can be conducted in the negotiation process of the discrete control part.

## REFERENCES

- [1] C. Kitts and M. Egerstedt, “Design, control, and applications of real-world multirobot systems [from the guest editors],” *IEEE Robot. Autom. Mag.*, vol. 15, no. 1, p. 8, Mar. 2008.
- [2] A. Khamis, A. Hussein, and A. Elmogy, “Multi-robot task allocation: A review of the state-of-the-art,” in *Cooperative Robots and Sensor Networks 2015*, A. Koubâa and J. R. Martínez-de Dios, Eds. Switzerland: Springer, 2015, pp. 31–51.
- [3] 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.
- [4] 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.
- [5] M. Kloetzer, C. Mahulea, and R. Gonzalez, “Optimizing cell decomposition path planning for mobile robots using different metrics,” in *Proc. Int. Conf. Syst. Th., Contr. Comput.*, Cheile Gradistei, Romania, Oct. 2015, pp. 565–570.
- [6] M. Pivtoraiko and A. Kelly, “Kinodynamic motion planning with state lattice motion primitives,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots. Syst.*, San Francisco, CA, USA, Sept. 2011, pp. 2172–2179.
- [7] P. Bhattacharya and M. L. Gavrilova, “Roadmap-based path planning - Using the Voronoi diagram for a clearance-based shortest path,” *IEEE Robot. Autom. Mag.*, vol. 15, no. 2, pp. 58–66, Jun. 2008.
- [8] T. Lai, F. Ramos, and G. Francis, “Balancing global exploration and local-connectivity exploitation with rapidly-exploring random disjoints-trees,” in *Proc. IEEE Int. Conf. Robot. Automata.*, Montreal, QC, Canada, May 2019, pp. 5537–5543.
- [9] L. Huang, M. Zhou, K. Hao *et al.*, “Multirobot cooperative patrolling strategy for moving objects,” *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 53, no. 5, pp. 2995–3007, May 2023.
- [10] L. Huang, M. Zhou, and K. Hao, “Non-dominated immune-endocrine short feedback algorithm for multi-robot maritime patrolling,” *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 1, pp. 362–373, Jan. 2020.
- [11] L. Huang, M. Zhou, K. Hao *et al.*, “A survey of multi-robot regular and adversarial patrolling,” *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 4, pp. 894–903, Jul. 2019.
- [12] Y. Zhou, H. Hu, Y. Liu *et al.*, “A real-time and fully distributed approach to motion planning for multirobot systems,” *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 12, pp. 2636–2650, Dec. 2019.
- [13] 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, Sept. 2011, pp. 3645–3652.
- [14] C. Liu, C.-W. Lin, S. Shiraishi *et al.*, “Distributed conflict resolution for connected autonomous vehicles,” *IEEE Trans. Intell. Veh.*, vol. 3, no. 1, pp. 18–29, Mar. 2018.
- [15] Y. Zhou, H. Hu, Y. Liu *et al.*, “Collision and deadlock avoidance in multirobot systems: A distributed approach,” *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 7, pp. 1712–1726, Jul. 2017.
- [16] ———, “A distributed method to avoid higher-order deadlocks in multi-robot systems,” *Automatica*, vol. 112, pp. 108 706:1–108 706:13, Feb. 2020.
- [17] L. E. Kavraki, P. Svestka, J.-C. Latombe *et al.*, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.
- [18] V. R. Desaraju and J. P. How, “Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees,” in *Proc. IEEE Int. Conf. Robot. Automata.*, Shanghai, China, May 2011, pp. 4956–4961.
- [19] H. Choset, K. M. Lynch, S. Hutchinson *et al.*, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. Boston, MA: MIT Press, 2005.
- [20] D. V. Dimarogonas, S. G. Loizou, K. J. Kyriakopoulos *et al.*, “A feedback stabilization and collision avoidance scheme for multiple independent non-point agents,” *Automatica*, vol. 42, no. 2, pp. 229–243, Feb. 2006.
- [21] 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.
- [22] J. Van den Berg, M. Lin, and D. Manocha, “Reciprocal velocity obstacles for real-time multi-agent navigation,” in *Proc. IEEE Int. Conf. Robot. Autom.*, Pasadena, CA, USA, May 2008, pp. 1928–1935.
- [23] Y. Chen, M. Cutler, and J. P. How, “Decoupled multiagent path planning via incremental sequential convex programming,” in *Proc. IEEE Int. Conf. Robot. Autom.*, Seattle, WA, USA, May 2015, pp. 5954–5961.
- [24] Z. Li, J. Deng, R. Lu *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.
- [25] M. Guo and D. V. Dimarogonas, “Multi-agent plan reconfiguration under local LTL specifications,” *Int. J. Robot. Res.*, vol. 34, no. 2, pp. 218–235, Feb. 2015.
- [26] 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.
- [27] P. Abichandani, G. Ford, H. Y. Benson *et al.*, “Mathematical programming for multi-vehicle motion planning problems,” in *Proc. IEEE Int. Conf. Robot. Automata.*, Saint Paul, MN, USA, May 2012, pp. 3315–3322.
- [28] M. Jager and B. Nebel, “Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots. Syst.*, vol. 3, Maui, HI, USA, Oct. 2001, pp. 1213–1219.
- [29] R. L. Moorthy, W. Hock-Guan, N. Wing-Cheong *et al.*, “Cyclic deadlock prediction and avoidance for zone-controlled AGV system,” *Int. J. Prod. Econ.*, vol. 83, no. 3, pp. 309–324, Mar. 2003.
- [30] M. P. Fanti, A. M. Mangini, G. Pedroncelli *et al.*, “Decentralized deadlock-free control for AGV systems,” in *American Contr. Conf.*, Chicago, IL, USA, Jul. 2015, pp. 2414–2419.
- [31] M. Boukens, A. Boukabou, and M. Chadli, “A real time self-tuning motion controller for mobile robot systems,” *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 1, pp. 84–96, Jan. 2019.
- [32] Q. Wu, L. Yu, Y.-W. Wang *et al.*, “LESO-based position synchronization control for networked multi-axis servo systems with time-varying delay,” *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 4, pp. 1116–1123, Jul. 2020.
- [33] T. Wang, X. Xu, and X. Tang, “Scalable clock synchronization analysis: A symmetric noncooperative output feedback tubes-MPC approach,” *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 6, pp. 1604–1626, Nov. 2020.
- [34] Y. Zhou, H. Hu, Y. Liu *et al.*, “A distributed approach to robust control of multi-robot systems,” *Automatica*, vol. 98, pp. 1–13, Dec. 2018.
- [35] J. L. Garriga and M. Soroudi, “Model predictive control tuning methods: A review,” *Ind. Eng. Chem. Res.*, vol. 49, no. 8, pp. 3505–3515, Mar. 2010.
- [36] M. Mehndiratta, E. Camci, and E. Kayacan, “Automated tuning of nonlinear model predictive controller by reinforcement learning,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Madrid, Spain, Oct. 2018, pp. 3016–3021.
- [37] R. Nebeluk and M. Ławryńczuk, “Tuning of multivariable model predictive control for industrial tasks,” *Algorithms*, vol. 14, no. 1, pp. 10: 1–15, Jan. 2021.
- [38] D. Morgan, S.-J. Chung, and F. Y. Hadaegh, “Model predictive control of swarms of spacecraft using sequential convex programming,” *J. Guid. Contr. Dynam.*, vol. 37, no. 6, pp. 1725–1740, Nov. 2014.
- [39] L. Vandenberghe. (2012) Convex Optimization: Modeling and Algorithms. [Online]. Available: <http://www.seas.ucla.edu/~vandenbe/shortcourses/mlss12-convexopt.pdf>
- [40] P. Joglekar and M. Woodside, “Evaluating the scalability of distributed systems,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 6, pp. 589–603, Jun. 2000.

- [41] Q. T. Dinh and M. Diehl, "Local convergence of sequential convex programming for nonconvex optimization," in *Recent Advances in Optimization and its Applications in Engineering*, M. Diehl, F. Glineur, E. Jarlebring *et al.*, Eds., 2010, pp. 93–102.
- [42] H. Zhao, Y. Wen, S. Wu *et al.*, "Dynamic evaluation strategies for multiple aircrafts formation using collision and matching probabilities," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 4, pp. 890–904, Apr. 2021.
- [43] T. Zhang, J. Xiao, L. Li *et al.*, "Toward coordination control of multiple fish-like robots: Real-time vision-based pose estimation and tracking via deep neural networks," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 12, pp. 1964–1976, Dec. 2021.



**Yuan Zhou** received his M.S. degree in computational mathematics from Zhejiang Sci-Tech University, Hangzhou, China, in March 2015 and Ph.D. degree in computer science from Nanyang Technological University, Singapore, in June 2019. He is currently a Research Fellow in the School of Computer Science and Engineering at Nanyang Technological University.

His research interests focus on the safety and security of autonomous unmanned systems, including multi-robot systems and autonomous vehicles. He has published about 40 research papers in various journals and conferences including IEEE Transactions on Systems, Man, and Cybernetics: Systems, IEEE Transactions on Fuzzy Systems, IEEE Transactions on Software Engineering, IEEE Transactions on Automation Science and Engineering, IEEE Transactions on Reliability, IEEE Symposium on Security and Privacy (S&P), International Conference on Robotics and Automation (ICRA), International Symposium on Software Testing and Analysis (ISSTA), and International Conference on Software Engineering (ICSE).



**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, Shaanxi, China, in 2003, 2005, and 2010, respectively.

He is currently a Full Professor at Xidian University. His research interests include discrete event systems and their supervisory control techniques, Petri nets, automated manufacturing systems, multimedia streaming systems, and artificial intelligence. In the aforementioned areas, he has over 100 publications

in journals, book chapters, and conference proceedings. Dr. Hu is an Associate Editor for IEEE Control Systems Magazine, IEEE Robotics & Automation Magazine, IEEE Transactions on Automation Science and Engineering, Journal of Intelligent Manufacturing, etc. He was on the Editorial Board of 10+ international journals. He was the recipient of many national and international awards including the Franklin V. Taylor Outstanding Paper Award from the IEEE SMC Society in 2010 and the finalists of the Best Automation Paper from the IEEE ICRA Society in 2013 and 2016, respectively.



**Gelei Deng** is currently a Ph.D. student at Nanyang Technological University, Singapore. He received his Bachelor's degree from Singapore University of Technology and Design in 2018. His research focuses on computer system security, robotic security, and security testing.



**Kun Cheng** received his BS degree in 2012 and Ph.D. degree in 2020, both from the School of Computer Science and Engineering, Beihang University, Beijing, China. His research interests include system virtualization, cloud computing, and safety-critical systems.



**Shang-Wei Lin** received his B.S. degree in Information Management from the National Chung Cheng University, Chiayi, Taiwan, in 2003 and received his Ph.D. degree in Computer Science and Information Engineering from the National Chung Cheng University, Chiayi, Taiwan, in 2010.

He is currently an Assistant Professor at the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests include formal methods, embedded systems, and cyber-physical systems.



**Yang Liu** graduated from National University of Singapore (NUS) with a Bachelor degree of Computer Science in 2005. In 2010, he obtained his PhD in Computer Science and continued with his postdoctoral work at NUS.

He is currently a Full Professor at the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research focuses on software engineering, security, cyber-physical systems, and formal methods. Particularly, he specializes in software verification using model checker, Process Analysis Toolkit (PAT).



**Zuohua Ding** received the M.S. degree in computer science and the Ph.D. degree in mathematics, both 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 of the Laboratory of Intelligent Computing and Software Engineering, Zhejiang Sci-Tech University, Hangzhou, China. His current research interests include system modeling, program analysis, service computing, software reliability prediction, and Petri nets. He has authored and co-authored over 120 papers.