Deadlock-Free Online Plan Repair in Multi-robot Coordination with Disturbances

Adem Coskun, Jason O'Kane, and Marco Valtorta
Department of Computer Science and Engineering, University of South Carolina
acoskun@email.sc.edu, {jokane, mgv}@cse.sc.edu

Abstract

Multirobot systems are increasingly deployed in environments where they interact with humans. From the perspective of a robot, such interaction could be considered a disturbance that causes a well-planned trajectory to fail. Previous approaches that modify trajectories in the presence of disturbances rearrange the order in which robots pass collision regions and other obstacles, in the laudable attempt to improve the average travel time for all robots. By doing so, however, deadlock may arise. In this paper, we provide a precise definition of deadlock using a graphical representation and prove some of its important properties. We show how to exploit the representation to detect the possibility of deadlock and to characterize conditions under which deadlock may not occur. We provide experiments in simulated environments that illustrate the potential usefulness of our theory of deadlock.

Introduction

In the near future, collaboration between autonomous robots and humans will increase. Even now, we share many work spaces with robots; for example, there are some multi-robot systems operating in warehouses and they may encounter humans. If the trajectories of the robots are generated jointly, then each robot must follow its own trajectory in a precise way so that all robots can reach their destination. It is more challenging to operate robots safely and efficiently in an environment where robots are required to avoid colliding with humans. In recent work, Coskun and O'Kane (Coskun and O'Kane 2019) utilized the control strategy developed by Čáp, and Gregoire, and Frazzoli (Čáp, Gregoire, and Frazzoli 2016) and modified it by flipping the order of passing the common collision area between robots when some disturbances in the environment make the robots stop, in order to reduce robot travel times. Our paper extends that line of work by providing a theoretical analysis of how to detect deadlock and avoid it under the condition of flipping.

In this paper, we will provide a precise definition of deadlock and a graph representation of the trajectories in the coordination space approach, a theoretical analysis of the feasibility and complexity of detecting deadlocks in the system when the label of the coordination space obstacle is subject to change, and a simulation demonstrating how to detect and avoid deadlocks in several environments.

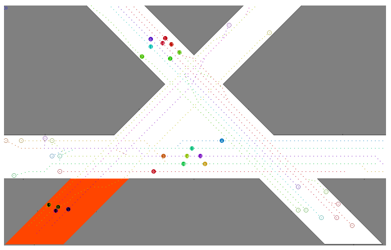


Figure 1: An environment with 20 robots. The dotted lines represent the planned trajectories of the robots. The robots in the red area are subject to a disturbance, which may be caused by interaction with humans or other objects and results in a delay in their progress.

In the remainder of the paper, we first review related work and give the formulation of the problem. Then, we propose our deadlock approach with a theoretical analysis, and describe our new algorithm to detect and avoid deadlocks. Finally, we present our simulation results, and conclude with some direction for future work.

Related Work

The problem of coordinating multiple robots in a shared environment can generally be categorized into two groups, namely *reactive* and *planning* approaches. In the reactive approaches, each robot follows its own shortest path, and collisions are resolved locally by observing the other robots' positions and velocities. The optimal reciprocal collision avoidance (ORCA) (Van Den Berg et al. 2011) formulation is used in practice due to its efficiency in calculating the velocity of the obstacles. However, reactive approaches do not guarantee that all robots reach their goal positions, so they are subject to having deadlock. In the planning approaches, all robots trajectories are generated by planning them before the robots start to execute their paths. These approaches guarantee that all robots reach their goal positions. However, their complexity increases exponentially with the number of coordinated robots. A configuration space (Lozano-Perez 1990) for a robot represents all points that the robot can reach, so planning for a robot reduces to finding a path between its start and goal positions in the configuration space. A configuration space for a multi robot system is the Cartesian product of the configuration spaces for each robot. In order to reduce the complexity of the planning approach, the path-velocity decomposition described in (Kant and Zucker 1986) is used. The decomposition consists of planning the path to avoid collisions with static obstacles, and planning the velocity to avoid collisions with dynamic obstacles. Such decomposition leads to the notion of *coordination space* originally proposed in (O'Donnell and Lozano-Pérez 1989), and extended to the geometry-based approach described in (Leroy, Laumond, and Siméon 1999). One of the most practical class of planning algorithms is that of prioritized algorithms, (Erdmann and Lozano-Perez 1987), because the trajectories of each robot are planned one after another instead of for all robots at once. A revised version of prioritized planning (Čáp et al. 2015) is used in our simulation to generate initial trajectories. In order to make sure that the system does not end up in deadlock, the robots must execute their trajectories in the same temporal sequence as planned. When robots cannot follow the planned trajectories because of delays due to disturbances, one can use the algorithm in (Čáp, Gregoire, and Frazzoli 2016) to allow limited changes to plans without introducing deadlock. The extension by (Coskun and O'Kane 2019) gives additional freedom to the robots to repair online their planned trajectories.

Problem Statement

In a planar environment $\mathcal{W} \subseteq \mathbb{R}^2$, each of n disc-shaped (with radius r) holonomic *robots*, indexed $1, \ldots, n$, moves, from a known *start position* to a known *goal position* in $\mathcal{W}$. Time is modeled as a series of discrete steps.

The robots' motions are initially planned by some multi-robot trajectory planner, which generates a family of feasible and jointly collision-free trajectories from the robots' start positions to their goal positions. Specifically, the planner generates a trajectory for each robot, of the form $\pi_i : \{1, \ldots, K_i\} \to \mathcal{W}$, in which K_i denotes the time step when robot i would reach its goal position when executing this nominal trajectory. Thus $\pi_i(k)$ denotes the position in $\mathcal{W}$ occupied by the center of robot i, when that robot has reached step k of its trajectory.

We are interested in scenarios wherein the robots cannot, for reasons unknown at planning time, always make progress along their trajectories. We refer to such a situation as a *disturbance*, meaning that the robot is unexpectedly barred from moving forward along its trajectory. This is modeled by a disturbance function, $\delta_i : \mathbb{N} \to \{0, 1\}$, under which robot i experiences a disturbance at time t if and only if $\delta_i(t) = 0$. The disturbances are generated by some random process unknown to the robots, which may vary across $\mathcal{W}$.

At each time step, each robot can elect to attempt to move forward along its trajectory or to remain intentionally motionless (presumably waiting to avoid a collision). We model this for each robot by an action function (or *policy*), $a_i : \mathbb{N} \to \{0, 1\}$. Here $a_i(t) = 1$ indicates that robot i does attempt to move along its trajectory at time t.

The actual progress of robot i along its trajectory at time t is represented as $x_i(t) \in \{1, \ldots, K_i\}$. Thus, at time t, the position within $\mathcal{W}$ of robot i is $\pi(x_i(t))$. Moreover, combining the semantics of the disturbance and action functions above, we obtain $x_i(t+1) = x_i(t) + a_i(t)\delta_i(t)$.

The goal is for each robot i to choose, at each time t, whether to move or wait (that is, $a_i(t) = 0$ or $a_i(t) = 1$), in such a way that each robot reaches its goal, without any collisions between robots, in minimal total time.

Summary of baseline approaches

This section reviews the approaches to this problem in both (Čáp, Gregoire, and Frazzoli 2016) and (Coskun and O'Kane 2019), on which this paper builds. Additional detail may be found in those original papers.

To help ensure that the robots, in spite of the disturbances, nonetheless do not collide, we utilize their *coordination spaces*. The coordination space for robots i and j is $C_{ij} = \{(k_i, k_j) \mid ||\pi_i(k_i) - \pi_j(k_j)|| \geq 2r\}$. The obstacle region for robots i and robot j is defined as $O_{ij} = \{1, \ldots K_i\} \times \{1, \ldots, K_j\} - C_{ij}$. The obstacle region O_{ij} can be partitioned into maximal connected regions, and represented as $O_{ij} = o_1^{ij} \cup \cdots \cup o_m^{(ij)}$. Each $o_k^{(ij)}$ is called a *coordination space obstacle* for robot i and robot j.

The movements of robot i and j can be represented as a path in the coordination space from $(1, 1)$ to (K_i, K_j) For each coordination space obstacle, the path must pass either *over* or *under* the obstacle. This is denoted as $\ell(o_k^{ij}) \in \{i, j\}$. If $\ell(o_k^{ij}) = j$, the path in the coordination space passes over the obstacle, which implies that in the workspace robot j passes that collision region first. Likewise, if $\ell(o_k^{ij}) = i$, the coordination space path travels under the obstacle and robot i goes through first.

Based on these obstacle labels (Čáp, Gregoire, and Frazzoli 2016), RMTRACK uses a policy equivalent to following action function:

$$a_i(t) = \begin{cases} 0 & \text{if } \exists j \neq i, \text{ s.t. } \exists k : \ell(o_k^{ij}) = j \text{ and} \\ & o_k^{ij} \cap (\{x_i(t)+1\} \times \{x_j(t), ..., K_j\} \neq \emptyset \\ 0 & \text{if } x_i(t) = K_i \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

The three cases of Equation 1 correspond to three distinct states for each robot:

- *Waiting (first case in Eq. 1):* If there exists a coordination space obstacle, o_k^{ij} for robot i and j with label j, so that $\ell(o_k^{ij}) = j$, then we check whether there is an intersection between the obstacle and the line from $(x_i(t) + 1, x_j(t))$ to $(x_i(t) + 1, K_j)$. The intersection indicates that robot j did not pass the obstacle yet, and robot i should therefore wait; thus the action variable is 0.
- *Finished (second case in Eq. 1):* When robot i reaches its destination, it should not move any further. The action variable then becomes 0.
- *Moving State (third case in Eq. 1):* If neither of the first two cases applies, then robot i is free to move, taking the action variable as 1.

As shown in (Čáp, Gregoire, and Frazzoli 2016), a system implementing Equation 1 can effectively overcome disturbances as the robots execute their nominal trajectories. However, that approach maintains (in our terminology) the same label for each obstacle as the nominal trajectories. In cases where disturbances are distributed non-uniformly across the environment, this can cause unnecessary delays. In response, (Coskun and O'Kane 2019) introduced two techniques called RMTRACK+TFF and RMTRACK+TFA (named for 'test flip fast' and 'test flip aggressive') that selectively modify some obstacle labels on-the-fly. For example, suppose robots i and j share a coordination space obstacle o_k^{ij}, with $\ell(o_k^{ij}) = i$. If robot i experiences a lengthy delay before moving past this obstacle, then robot j must wait when it reaches the obstacle o_k^{ij}, until robot i passes it. In such a case, RMTRACK+TFF and RMTRACK+TFA estimate the expected travel times for robot i and robot j with two different approaches, and predict whether changing the label of o_k^{ij} —allowing j to proceed immediately— would reduce total travel times for the robots i and j. If so then the label is changed. The two variants TTF and TFA differ in their tradeoffs between the computation time invested in deciding whether to 'flip' an obstacle and the acuity in detecting situations where such a flip would be beneficial.

Conditions for deadlock-free executions

Though RMTRACK+TFF and RMTRACK+TFA can effectively modify the coordination plan (by judiciously changing obstacle labels) to improve overall efficiency, there are circumstances in which those changes can lead to deadlock conditions, wherein none of the robots can make progress. In this section, we provide a precise definition of deadlock in this context, and prove that, under certain reasonable but not universal conditions, deadlock does not occur. These results then form the foundation for the deadlock-free coordination scheme we introduce in the next section.

We begin with a definition of deadlock. The three different states —waiting, moving, and finished— for each robot defined in Eq. 1 form the basis of the definition.

Definition 1 (deadlock)**.** The system has a *deadlock* if at least one robot is in the waiting state and no robots are in the moving state.

Our concern is to understand the conditions under which deadlocks can occur, to ensure that those conditions do not arise. To that end, we first introduce *collision segments* in coordination spaces.

Definition 2 (collision segment)**.** For a given coordination space obstacle o_k^{ij}, the *collision segment* c_k^{ij} for that obstacle is the set of indices in $\{1, \ldots, K_i\}$ along the path for robot i between the path step at which robot i reaches that obstacle, denoted $s(c_k^{ij})$, and the path step at which robot i clears that obstacle, denoted $f(c_k^{ij})$. That is, $c_k^{ij} = \{s(c_k^{ij}), \ldots, f(c_k^{ij}) - 1\}$.

Figure 2 shows some example collision segments. Such segments are important because they capture the structure of how the movements of the various robots affect each other. Notice in particular that the s and f functions induce a partition of the path of robot i into segments delimited by the start

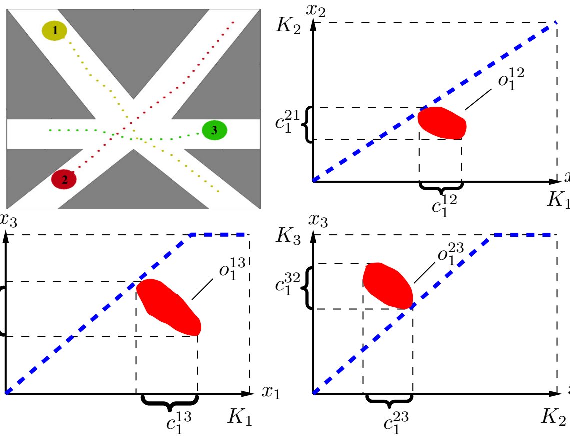


Figure 2: Three robots that have pairwise collisions in the environment are depicted on the top left. The coordination spaces, C_{12}, C_{13}, and C_{23}, are depicted on the top right, on the bottom left, and on the bottom right, respectively. The collision segments corresponds to the obstacles shown in the coordination spaces.

and end points of all of its collision segments with all other robots. Using this partition, we can express the relationship between paths of the robots and labels of the obstacles.

Definition 3 (segment graph)**.** The *segment graph* is a directed graph containing vertices corresponding to the maximal path segments for each robot delimited by the start and the finish of its collision segments with all other robots, denoted $v_1^{(i)}, v_2^{(i)}, \ldots, v_{m_i}^{(i)}$ for each robot i; edges called *path sequence edges* between each successive pair of vertices $v_k^{(i)}$ and $v_{k+1}^{(i)}$; and for each coordination space obstacle o_k^{ij} with label i, an edge called an *obstacle-label edge* from the vertex for robot i corresponding to the final path segment overlapping o_k^{ij} to the vertex for robot j corresponding to the first path segment overlapping o_k^{ji}.

Figure 3 shows the segment graph for the scenario depicted in Figure 2. The intuition is that each edge in a segment graph describes a constraint wherein one segment of some robot's path must be completed before another segment can begin. Path sequence edges encode the constraint that each robot must execute its path sequentially; obstacle label edges encode the waiting behaviour required by the first case in Equation 1.

It will be convenient later to refer to the starting and ending points of the path segment for each vertex. We (re-)use the letters s and f for this purpose, so that vertex $v_k^{(i)}$ in the segment graph corresponds to the range of steps $s(v_k^{(i)}), \ldots, f(v_k^{(i)})$ within the path for robot i. Notice that, in general, the finish of one vertex is immediately before the start of another: $f(v_k^{(i)}) = s(v_{k+1}^{(i)})$.

In addition, we write $V(o_k^{ij}) = \{v_p^{(i)}, v_{p+1}^{(i)}, \ldots v_{p+q}^{(i)}\} \subset V$ for the set of robot i vertices containing obstacle o_k^{ij}.

As the robots execute their paths, they move in sequence

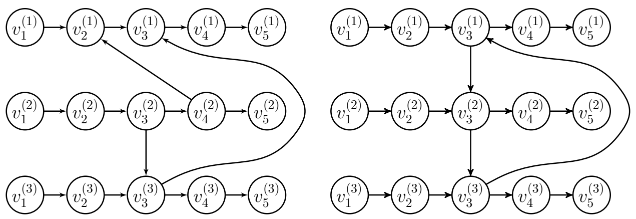


Figure 3: [left] The segment graph for the three robots depicted in Figure 2. There are five vertices for each robot because each robot has two collisions with other robots. The horizontal edges represent the path sequence edges for each robot, and the other three edges represent obstacle label edges for each obstacle. Since the coordination space obstacle o_1^{23} has label $\ell(o_1^{23}) = 2$, $V(o_1^{23}) = \{v_2^{(2)}, v_3^{(2)}\}$, and $V(o_1^{32}) = \{v_3^{(3)}, v_4^{(3)}\}$, there is an edge from $v_3^{(2)}$ to $v_3^{(3)}$. [right] The segment graph when the label of the obstacle, o_1^{12} changes from 2 to 1, indicating that robot 1 passes the obstacle before robot 2.

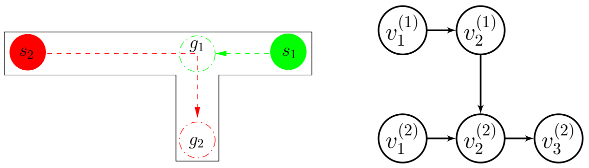


Figure 4: If robot 1 reaches its goal position, g_1, before robot 2 clears the intersection, then the system has a deadlock. Even though the environment (left) has a deadlock, the corresponding segment graph (right) does not have a cycle.

through the segments of their paths. Thus, we can refer to the *present vertex* in the segment graph for each robot. Any vertex corresponding to a path segment that the robot has not yet begun to execute is a called a *future vertex* for that robot. The subgraph of the segment graph induced by the present and future vertices is called the *active subgraph*.

The next definition is needed to connect segment graphs to the possibility of deadlocks in the future.

Definition 4. A set of trajectories is *non-conflicting* if, for every pair of i, j of distinct robots, for all $1 \leq k \leq K_i$, we have $||\pi_i(k) - \pi_j(1)|| \geq 2r$, and $||\pi_i(k) - \pi_j(K_j)|| \geq 2r$.

That is, when we have non-conflicting trajectories, the path for each robot is disjoint from the start and goal positions of all other robots. Figure 4 shows an example of conflicting trajectories and the corresponding segment graph.

Now we can show that cycles in the segment graph correspond to potential deadlocks.

Theorem 1. If the system has non-conflicting trajectories and is in a deadlock, then the active subgraph has a cycle.

Proof. Definition 1 ensures that at least one robot is in the waiting state. Let i denote the index of this robot, and let $v_p^{(i)}$ denote its present vertex. Because of Definition 3, there must therefore be at least one edge incoming to $v_{p+1}^{(i)}$, i.e. the vertex corresponding to the next segment for robot i from a future vertex of some other robot. Let j denote the index of this other robot. Robot j cannot be in the moving state because the system is in a deadlock. Moreover, robot j cannot be in the finished state because being in the finished state, in a system with non-conflicting trajectories, does not prevent any other robots —robot i in particular— from moving. Thus, robot j is in the waiting state. Because robot j is waiting, there must be another incoming obstacle-label edge from another robot's future vertex to a future vertex of the robot j, and so on. This produces an infinite sequence of vertices in the graph, each connected by a directed edge to its predecessor in the sequence. Since the number of robots is finite, the vertices in this sequence cannot all be distinct. Therefore, the sequence must eventually repeat, forming a cycle in the active subgraph. □

Theorem 2. If the active subgraph of a system has a cycle, then the system has a deadlock.

Proof. It is obvious that if the system has one robot, then the segment graph has only path sequence edges, so a cycle cannot exist. Also, a cycle cannot exist when there are only two robots. Note that there must be only one incoming or one outgoing obstacle-label edge between two robots' vertices for each coordination space obstacle, so those obstacle-label edges do not lead to a cycle.

Now, assume that there are at least three robots in the system and that the active subgraph has a cycle between robot i, robot j, ... , and robot k, whose present vertices are $v_p^{(i)}, v_q^{(j)}, \ldots, v_r^{(k)}$, respectively.

Consider a cycle between future vertices of the robots in the segment graph: $v_{p+1}^{(i)} \to v_{q+1}^{(j)} \to \ldots \to v_{r+1}^{(k)} \to v_{p+1}^{(i)}$

Since there is an obstacle-label edge from robot k to robot i, robot i cannot start the future vertex, $v_{p+1}^{(i)}$ before robot k finishes its future vertex, $v_{r+1}^{(k)}$.

Similarly, there is also an obstacle-label edge from robot i to robot j, so robot j cannot start the future vertex, $v_{q+1}^{(j)}$ before robot i finishes its future vertex, $v_{p+1}^{(i)}$.

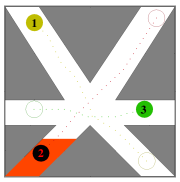
Therefore, no robots in the cycle start their future vertex and stay in the waiting state, so the system has a deadlock. □

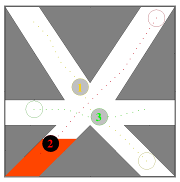
Corollary 1. If, in a system with non-conflicting trajectories, the active subgraph contains no cycles, then the system is not in a deadlock.

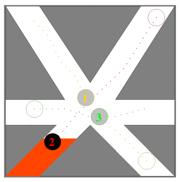
This last result is of particular interest because, since the initial trajectories are collision free, the only way a cycle can be introduced in the segment graph is by modifying one of the obstacle labels, which in turn rewires one of the obstacle label edges. This idea underlies the deadlock free coordination approach in the next section.

Moving beyond that observation, we now also provide a sufficient condition, which results a deadlock-free system, even in cases were the obstacle labels can be freely modified. We first examine the disjointedness of collision segments.

Theorem 3. If the collision segments are disjoint, then each set $V(o_k^{ij})$ contains only a single vertex.


(a)


(b)


(c)

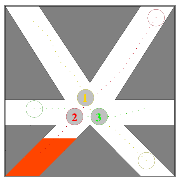

(d)

Figure 5: An example deadlock configuration.

Proof. Suppose $|V(o_k^{ij})| \neq 1$. The existence of more than one vertex in the set $V(o_k^{ij})$ indicates there exists at least one more collision segment reached or cleared by robot i before robot i clears o_k^{ij}. This contradicts the assumption that collision segments do not intersect, so $|V(o_k^{ij})| = 1$. $\square$

The existence of these singleton vertex-segment sets is of interest because it allows us to demonstrate that the corresponding segment graph does not have a cycle.

Theorem 4. If the collision segments are disjoint, then the segment graph does not contain a cycle.

Proof. Suppose the segment graph has a cycle such that $v_p^{(i)} \to v_q^{(j)} \to \cdots \to v_r^{(k)} \to v_p^{(i)}$ with $V(o_l^{ij}) = \{v_p^{(i)}\}$, $V(o_l^{ji}) = \{v_q^{(j)}\}, \ldots, V(o_m^{ki}) = \{v_r^{(k)}\}$, and $V(o_m^{ik}) = \{v_p^{(i)}\}$.

Since $V(o_l^{ij}) = \{v_p^{(i)}\}$, $s(v_p^{(i)})$ must be the number of path steps to reach obstacle o_l^{ij}. However, since $V(o_m^{ik})$ contains only a single vertex, $s(v_p^{(i)})$ is also the number of path steps to reach the obstacle o_m^{ik}. This contradicts the assumption that the collision segments are disjoint. Therefore the segment graph does not have a cycle. $\square$

Finally, viewing Theorem 4 in light of Corollary 1, one sees conditions which, though not directly leveraged in the framework of this paper, may nonetheless be useful when designing multi-robot trajectory planners. If the planner generates paths that meet that condition — in this context, the requirement is that intersection points between paths must be at least distance $2r$ apart from each other, then deadlocks can be avoided, even if a method such as RMTRACK+TFF or RMTRACK+TFA modifies obstacle labels.

Algorithm Description

The multi-robot trajectory planner in (Čáp et al. 2015) generates the initial trajectories for the robots. When this offline trajectory planner generates the plan, the disturbances are unknown and the probability of having disturbances is non-uniform in the environment.

When a robot reaches the obstacle before another one reaches it due to disturbances, and the label of the obstacle belongs to the other robot, then the robot must stay in the waiting state. There are two approaches to decide whether changing the label is beneficial or not as described in (Coskun and O'Kane 2019).

Briefly, the RMTRACK+TFF approach only calculates expected travel time to clear the obstacle for the robot in the waiting state, and expected travel time to reach the obstacle for the robot, having disturbances. If the first calculated value is less than the second one, the robot in the waiting state can clear the obstacle before the other one reaches it, so the label is changed; otherwise the robot in the waiting state does not move until the other one clears the obstacle. On the other hand, the RMTRACK+TFA approach calculates four different expected travel times taking into account both robots' movement. It first calculates the expected travel times to clear the obstacle for both robots with the current label. Then, it assumes the label is changed, and recalculates the expected travel times to clear the obstacle for both robots. If the sum of the last two expected travel times is less than the sum of the first two expected travel times, changing the label is overall beneficial because the total travel time is reduced, so the label is changed.

The main contribution of the proposed algorithm is that it allows testing to determine whether changing an obstacle label leads to a deadlock. The algorithm first generates the segment graph by using a Java library, JGraphT[1]. Then, if the segment graph detects a cycle with the updated label, the algorithm does not change that label. For example, let us recall the example depicted in Figure 2. There are three robots, and there are three pairwise obstacles between each robot. According to the label of the obstacles, robot 2 should pass the obstacle before robot 1 passes, as shown in C_{12}, on the top right of Figure 2. However, robot 2 is subject to more disturbance than the others, and robot 1 reaches the obstacle before robot 2 as depicted on Figure 5b. There are two scenarios for Robot 1, staying in the waiting state until robot 2 clears the obstacle, or flipping the label of obstacle and continuing to execute its path. If robot 1 changes the label, then it can move, but it needs to stop to avoid collision with robot 3, which is in the waiting state by waiting for robot 2, depicted on Figure 5c. Then, when robot 2 reaches the obstacle, a deadlock ensues as shown in Figure 5d. In order to avoid this deadlock, when robot 1 needs to change the label between robot 2, the segment graph is updated, and checked for the occurrence of a cycle. If a cycle occurs as shown on the right of Figure 3, then the algorithm does not change the label, and robot 1 stays in the waiting state so that the system does not have a deadlock.

Experimental Results

We have extended the original code for RMTRACK simulation[2] by adding flipping algorithms and our new deadlock detection method, and implemented it in Java with an Ubuntu 20.04 computer and a 2.9GHz processor.

[1]https://jgrapht.org/

[2]https://github.com/mcapino/rmtrack

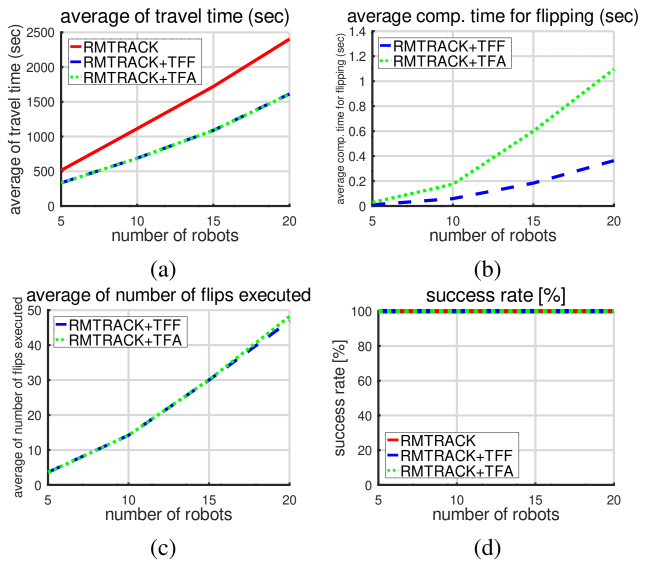


Figure 6: Experimental Results

The environment for the experiments in this paper is shown in Figure 1. The probability of having disturbance in the red area is 0.85, and for the rest of the environment, it is 0.05. For each number of robots (5, 10, 15, and 20), we have 10 different randomly generated start and goal positions.

The average travel times of RMTRACK and two flipping algorithms are shown in Figure 6a. The flipping algorithms reduced the travel times and avoided deadlock, so that all robots reached their goal positions as shown in Figure 6d.

The average computation time for RMTRACK+TFA is longer than the average computation time for RMTRACK+TFF as shown in Figure 6b because RMTRACK+TFA calculates the expected travel times by considering the movements of both robots with or without flipping the label of the obstacle. Also, note that computation time in general is small when compared with travel time.

The number of flips is almost the same for two flipping algorithms in Figure 6c because the collisions between two robots may happen only in a small area of their trajectories. On the other hand, if two robots travels in a long and narrow passage, then collisions may happen in a large area of their trajectories, and we observed (not shown) that RMTRACK+TFA performs better than RMTRACK+TFF.

Simulation videos and detailed segment graph examples, including an environment in which deadlock occurs when running the new algorithms, are provided in a repository.[3]

Conclusion

We provided a theoretical analysis of the conditions under which flipping algorithms lead to deadlock, and provided conditions for deadlock-free environment even in the presence of flips. Our simulation results complement the theoretical ones by showing that flipping algorithms significantly reduce robot travel times when the difference in disturbance probability between different areas is large. However, since flipping algorithms may lead to deadlock in some situation, we have proposed an algorithm based on the segment graph data structure to detect and avoid deadlocks before flipping, thus combining the efficiency of flipping algorithms with a theoretical guarantee of deadlock avoidance. A reviewer suggested that we investigate the algorithms described in (Ma, Kumar, and Koenig 2017; Berndt et al. 2020), which seem to be related to ours. We plan to do that in the near future. In future work, we also plan to use segment graphs to detect and avoid deadlocks in decentralized approaches to multi-robot coordination.

[3]`https://www.dropbox.com/sh/h7bj8pm0ic53puq/AACN6WeJO0H7_rXSmpyYL19ja?dl=0`

References

[Berndt et al. 2020] Berndt, A.; Van Duijkeren, N.; Palmieri, L.; and Keviczky, T. 2020. A feedback scheme to reorder a multi-agent execution schedule by persistently optimizing a switchable action dependency graph. *arXiv preprint arXiv:2010.05254*.

[Čáp et al. 2015] Čáp, M.; Novák, P.; Kleiner, A.; and Seleckỳ, M. 2015. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE Transactions on Automation Science and Engineering* 12(3):835–849.

[Čáp, Gregoire, and Frazzoli 2016] Čáp, M.; Gregoire, J.; and Frazzoli, E. 2016. Provably safe and deadlock-free execution of multi-robot plans under delaying disturbances. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5113–5118. IEEE.

[Coskun and O'Kane 2019] Coskun, A., and O'Kane, J. M. 2019. Online plan repair in multi-robot coordination with disturbances. In *International Conference on Robotics and Automation (ICRA)*, 3333–3339. IEEE.

[Erdmann and Lozano-Perez 1987] Erdmann, M., and Lozano-Perez, T. 1987. On multiple moving objects. *Algorithmica* 2(1):477–521.

[Kant and Zucker 1986] Kant, K., and Zucker, S. W. 1986. Toward efficient trajectory planning: The path-velocity decomposition. *The International Journal of Robotics Research* 5(3):72–89.

[Leroy, Laumond, and Siméon 1999] Leroy, S.; Laumond, J.-P.; and Siméon, T. 1999. Multiple path coordination for mobile robots: A geometric algorithm. In *IJCAI*, volume 99, 1118–1123.

[Lozano-Perez 1990] Lozano-Perez, T. 1990. Spatial planning: A configuration space approach. In *Autonomous Robot Vehicles*. Springer. 259–271.

[Ma, Kumar, and Koenig 2017] Ma, H.; Kumar, T. S.; and Koenig, S. 2017. Multi-agent path finding with delay probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

[O'Donnell and Lozano-Pérez 1989] O'Donnell, P. A., and Lozano-Pérez, T. 1989. Deadlock-free and collision-free coordination of two robot manipulators. In *1989 IEEE International Conference on Robotics and Automation*, 484–489. IEEE Computer Society.

[Van Den Berg et al. 2011] Van Den Berg, J.; Guy, S. J.; Lin, M.; and Manocha, D. 2011. Reciprocal n-body collision avoidance. In *Robotics Research*. Springer. 3–19.