Departure and Conflict Management in Multi-Robot Path Coordination

Puttichai Lertkultanon, Yang Jingyi, Hung Pham, and Quang-Cuong Pham
ATMRI, School of Mechanical and Aerospace Engineering
Nanyang Technological University, Singapore
Email: cuong.pham@normalesup.org

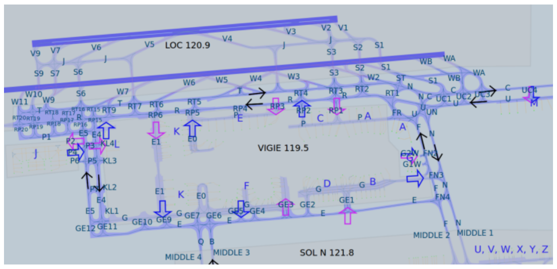
***Abstract*—This paper addresses the problem of multi-robot path coordination, considering specific features that arise in applications such as automatic aircraft taxiing or driver-less cars coordination. The first feature is departure events: when robots arrive at their destinations (e.g. the runway for take-off), they can be removed from the coordination diagram. The second feature is the "no-backward–movement" constraint: the robots can only move forward on their assigned paths. These features can interact to give rise to complex conflict situations, which existing planners are unable to solve in practical times. We propose a set of algorithms to efficiently account for these features and validate these algorithms on a realistic model of Charles de Gaulle airport.**

I. Introduction

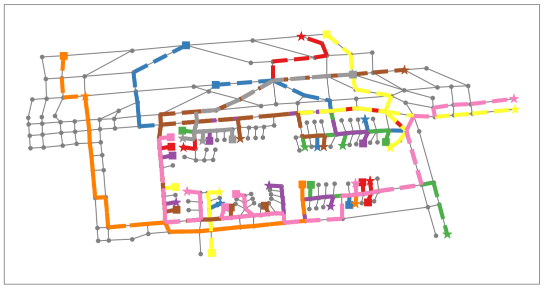
Consider n robots whose tasks are to move from their initial positions $\boldsymbol{p}_i^{\text{start}}$ to their respective goal positions $\boldsymbol{p}_i^{\text{goal}}$, where $i \in \{1, 2, \ldots, n\}$. Suppose furthermore that their *paths* from $\boldsymbol{p}_i^{\text{start}}$ to $\boldsymbol{p}_i^{\text{goal}}$ are fixed and do not involve any collision with the environment. Finding the coordinated motions of the robots to move along their paths while avoiding mutual collisions is a classical problem robotics, known as the *multi-robot path coordination* problem.

This problem arises in many contexts. In some multi-robot motion planning instances, the unconstrained problem (i.e. when the paths are not fixed) is intractable, in particular because of the high dimensionality of the problem itself, such that the only practical solution is to *decouple* it into two steps [1], [2], [3]: (i) find the paths for each robot independently; (ii) solve the path coordination problem with the paths found in (i) being fixed. In some other applications, the robots have to move along predefined "tracks", e.g. driver-less cars on city streets or automatic aircraft taxiing on airport tracks. Here, the constrains of "staying on the track" makes unconstrained motion planning inapplicable. A more natural solution consists of (i) finding the paths for each robot independently using, e.g. graph search; (ii) solving the path coordination problem with the paths found in step (i).

In this paper, we consider the multi-robot path coordination problem with automatic aircraft taxiing as an intended application (see Fig. 1 for an application scenario at Charles de Gaulle (CDG) airport in Paris, France). This application displays two specific features. The first feature is departure events: when robots arrive at their destinations (e.g. the runway for take-off), they can be removed from the coordination diagram, resulting in a change of the structure of the coordination



(a)



(b)

Fig. 1: (a) An actual map of the Charles de Gaulle (CDG) airport in Paris, France. (b) A simplified map of the airport used in the simulations presented in this paper. The figure includes paths of 25 aircraft, shown in different colors. Squares indicate start positions and stars indicate goal positions. A video of a coordination solution of 25 aircraft computed by the proposed planner can be found at `https://youtu.be/hfJeUKpeeD0`.

problem, such as its dimensionality. The second feature is the "no-backward-movement" constraint: the robots can only move forward on their assigned paths. This constraint also arises in driver-less cars applications: while reversing a car is possible, it should be avoided in normal driving. Furthermore, these features can interact to give rise to complex conflict situations, which existing planners are unable to solve in practical times. The objective of this paper is to develop a set of algorithms to address (a) departure events, and (b) the "no-backward-movement" constraint in the multi-robot path coordination problem. These algorithms significantly improve planning time and make coordinated motion planning suitable for real-time taxiing applications.

The rest of the paper is organized as follows. In Section II, we briefly review the multi-robot motion planning problem and some of its applications. In Section III, we recall the main concepts of path coordination. In Section IV, we present our main contributions, namely, a set of new algorithms to handle departure and conflicts, which are specifically caused by the "no-backward-movement" constraint. In Section V, we present simulation results based on a realistic model of Charles de Gaulle airport. Finally, in Section VI, we discuss the advantages and drawbacks of our approach and sketch some future research directions.

II. Literature Review

Approaches to the problem of multi-robot motion planning can be classified as *centralized* or *decentralized*. In the centralized approach, a central computer receives position information, computes the motion plans, and sends the commands to all robots. On the other hand, in the decentralized approach, each robot computes its own movement based on the information at its disposal.

In the context of aircraft taxiing, the centralized approach is currently in use in most airports (air traffic controllers play the role of the central computer), which prompts us to focus on this approach in this paper. Within the centralized approach, one can further distinguish *decoupled* methods, which we have mentioned previously, from *non-decoupled* methods, where trajectories (paths *and* timings) are computed in a single step.

A. Non-Decoupled Methods

Applying classical robot motion planning methods, such as probabilistic roadmap (PRM) [4] or rapidly-exploring random tree (RRT) [5], directly to the product configuration space is highly inefficient. Therefore, most non-decoupled methods are found in the operation research literature, and are based in particular on Mixed Integer Linear Programming (MILP).

In [6], [7], Marín formulated the problem as a multi-commodity flow network model, which is then solved via MILP. In particular, a planning horizon is first defined and then discretized into a number of intervals with the horizon often lasts 30 min while the intervals is approximately 30 s. Then a spatial-time graph is constructed from the original spatial graph by creating multiple instances of each node corresponding to the number of time interval. The author then solved the resulting network flow problem with constraints via CPLEX. This formulation has the ability to consider general objective function. Note that recent developments on Lagrangian decomposition [7] yield significant improvements in terms of computation time.

A different formulation that also utilizes MILP was reported in [8]. Here, no time-discretization is required. Instead, binary variables indicate the order with which aircraft pass through a given landmark. The authors reported that their implementation can plan motions for 30 aircraft in a reasonable amount of time (less than 100 s), but computation time increased significantly (more than 1000 s) for 50 aircraft.

B. Decoupled Methods

In [1], O'Donnell and Lozano-Pérez were the first to introduce the idea of coordination diagram, which encodes the relative motions of the robots on their assigned paths (see Section III-A for more details). The authors then proposed to compute the obstacle regions in the coordination diagram explicitly via discretization and subsequently to find a collision-free path by a greedy algorithm.

In [3], Siméon *et al.* considered the problem of multiple mobile vehicles moving along straight lines or arcs. This particular path geometry allows them to efficiently compute the *bounding box* representation of the obstacle regions in the coordination diagram.

In [9], Peng and Akella considered the problem of time-optimal path coordination subject to velocity and acceleration constraints. The authors formulated the problem using Mixed Integer Nonlinear Programming and were able to compute sub-optimal but feasible coordinations.

In [2], Svetska and Overmars introduce the idea of roadmap coordination. Instead of moving on an assigned path, a robot can move within a roadmap. The coordination diagram then becomes a *composite roadmap*. This can be seen a generalization of both the non-decoupled and decoupled methods: if the roadmaps are infinitely dense, then roadmap coordination is equivalent to the former approach; if each roadmap contains only one path, then roadmap coordination is equivalent to the latter. Next, using roadmap coordination, it is possible to decompose the global coordination problem into several connected components.

More recently, in [10], Solovey *et al.* used RRT to find collision-free paths in the coordination diagram and obtained promising results in terms of computation time.

C. Specificity of Our Approach

Our approach is inscribed within the coordination diagram framework. In particular, as in [10], we use RRT to find collision-free paths in the coordination diagram. Our specificity consists of the departure and conflict management features. As we shall demonstrate, taking these features into account significantly improves the performance with respect to using a vanilla motion planner. Note that the development presented here can be easily combined with existing decoupled planning heuristics (e.g. roadmap coordination [2]).

In absence of standardized test cases, it is difficult to establish quantitative comparisons with the non-decoupled approaches based on MILP discussed previously. Indeed, the hardness of a motion planning instance not only depends on the number of aircraft, but more fundamentally, on the existence of "narrow passages". Our problem instances are associated with a large number of shared tracks and potential deadlocks, yielding particularly narrow passages in the coordination diagram, which are hard to overcome without the proposed conflict management.

III. Background: Multi-Robot Path Coordination without Backward Movements

A. Coordination Diagram

Consider n robots traveling on n given paths $P_1, P_2, \dots, P_n$ with path lengths of $s_1^{\text{goal}}, s_2^{\text{goal}}, \dots, s_n^{\text{goal}}$, respectively. The positions of the robots on their respective paths can be given by an n-tuple $\boldsymbol{q} = (s_1, s_2, \dots, s_n)$, where $s_i \in [0, s_i^{\text{goal}}], 1 \leq i \leq n$. Thus, one can represent this vector, which we will refer to as a *configuration*, as a point in an n-orthotope $\mathscr{C} := [0, s_1^{\text{goal}}] \times [0, s_2^{\text{goal}}] \times \dots \times [0, s_n^{\text{goal}}]$, also called the *coordination diagram* (CD) [1], [2], [3].

As two paths P_i and P_j may intersect or even share some track segments, robots i and j may collide with each other while traversing their paths. For simplicity, robots are assumed to be identical and are represented as disks of radius $r/2$. One can thus define the *obstacle set* in $\mathscr{C}$ as

$$\mathscr{O} = \{\boldsymbol{q} \mid \exists i, j, \|\boldsymbol{p}_i(s_i) - \boldsymbol{p}_j(s_j)\|_2 \leq r\}, \qquad (1)$$

where $\boldsymbol{p}_i(s)$ is the 2D physical position of the robot i when it has traveled a distance s along its path.

The problem of path coordination—finding the motions for all robots along their paths so that they (i) start from the origin of their paths; (ii) eventually reach the end of their paths; and (iii) never collide with one another—can now be cast as a classical motion planning problem: find

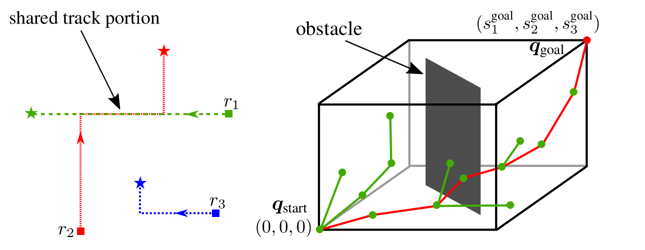

shared track portion
r_1
r_2
r_3
(a)
obstacle
$(s_1^{\text{goal}}, s_2^{\text{goal}}, s_3^{\text{goal}})$
$\boldsymbol{q}_{\text{goal}}$
$\boldsymbol{q}_{\text{start}}$
$(0,0,0)$
(b)

Fig. 2: (a) Three robot paths in the physical map. Squares are the start positions and Stars are the goal positions. Note that robots 1 and 2 share a common segment of their paths, on which they travel in opposite directions. (b) The corresponding coordination diagram. The obstacle induced by the shared path segment is depicted in dark gray. Note that for simplicity, the collision radius in this case was supposed to be very small. Otherwise, the obstacle would be thicker. The search tree is depicted in green and the solution path is highlighted in red.

a collision-free continuous path between $(0, 0, \ldots, 0)$ and $(s_1^{\text{goal}}, s_2^{\text{goal}}, \ldots, s_n^{\text{goal}})$ in the CD. Fig. 2 shows an example of coordination diagram and a search tree constructed for a problem with three robots.

The classical motion planning problem can be solved by any existing geometric motion planning algorithms [11]. Here we use a rapidly-exploring random tree (RRT) planner [5]. Briefly, RRT iteratively grows a tree rooted at the start configuration $\boldsymbol{q}_{\text{start}} := (0, 0, \ldots, 0)$. At each planning iteration, a random configuration $\boldsymbol{q}_{\text{rand}}$ is sampled within the free space $\mathscr{C} \setminus \mathscr{O}$. Next, one determines amongst the existing tree vertices, the vertex $\boldsymbol{q}_{\text{near}}$ that is the nearest neighbor to $\boldsymbol{q}_{\text{rand}}$, according to some distance function dist. Finally, one tries extending the tree from $\boldsymbol{q}_{\text{near}}$ towards $\boldsymbol{q}_{\text{rand}}$, creating thereby a new tree vertex $\boldsymbol{q}_{\text{new}}$. The algorithm terminates when either the time limit is reached or the goal configuration $\boldsymbol{q}_{\text{goal}} := (s_1^{\text{goal}}, s_2^{\text{goal}}, \ldots, s_n^{\text{goal}})$ can be connected to the tree. For more details and variations, the reader is referred to [5].

Collision Checking: When extending the tree from $\boldsymbol{q}_{\text{near}}$ towards $\boldsymbol{q}_{\text{rand}}$, one needs to check if the segment $(\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{rand}})$ [1] is collision-free. For this, one checks whether all discretized configurations $\boldsymbol{q}$ along the segment are collision-free.

Consider a configuration $\boldsymbol{q} = (s_1, s_2, \ldots, s_n)$. To check whether $\boldsymbol{q}$ is collision-free, one puts all the robots to the physical positions defined by the respective $s_i, i \in \{1, 2, \ldots, n\}$. Then one checks for all pairs (i, j) whether the robots i and j are in mutual collision, according to Eq. (1). Note that the obstacle set is thus not necessarily explicitly constructed in the coordination diagram.

B. Motion Planning without Backward Movements

Consider a straight path from $\boldsymbol{q} = (s_1, s_2, \ldots, s_n)$ to $\boldsymbol{q}' = (s'_1, s'_2, \ldots, s'_n)$ in $\mathscr{C}$. The "no-backward-movement" constraint is equivalent to the monotonicity of the path. Therefore, the path satisfies the "no-backward-movement" constraint *if and only if* $\boldsymbol{q} \preceq \boldsymbol{q}'$, where $\preceq$ is a component-wise inequality between two vectors.

[1]In case $\mathrm{dist}(\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{rand}}) > \varepsilon$, where ε is the maximum extension distance, one considers instead the segment $(\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{close}})$, where $\boldsymbol{q}_{\text{close}}$ is the configuration on the segment $(\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{rand}})$ at distance ε from $\boldsymbol{q}_{\text{near}}$.

To enforce this constraint in RRT, one should attempt extension from $\boldsymbol{q}_{\text{near}}$ to $\boldsymbol{q}_{\text{rand}}$ only when $\boldsymbol{q}_{\text{near}} \preceq \boldsymbol{q}_{\text{rand}}$. A simple way to implement this filter is to modify the distance function (used in the nearest neighbor search) as

$$\mathrm{dist}(\boldsymbol{q} \to \boldsymbol{q}') = \begin{cases} \|\boldsymbol{q} - \boldsymbol{q}'\| & \text{if} \quad \boldsymbol{q} \preceq \boldsymbol{q}' \\ +\infty & \text{otherwise} \end{cases} \tag{2}$$

such that any $\boldsymbol{q} \npreceq \boldsymbol{q}_{\text{rand}}$ will never be selected as the nearest neighbor of $\boldsymbol{q}_{\text{rand}}$.

IV. Departure and Conflict Management

A. Departure Management

We say that a robot *departs* from the CD when it reaches its goal position. This terminology is in line with the aircraft taxiing context, where aircraft indeed depart (take off) when they reach their goal positions (the runway) or wheel-locked in the gate where it would not block the way of other aircraft. Once a robot has departed, it "disappears" and is no longer taken into account in collision checks, thereby simplifying the motion planning problem.

In the CD $\mathscr{C}$, a departure of a robot corresponds to the search tree reaching a configuration $\boldsymbol{q}$ on an $n-1$-dimensional *face* of $\mathscr{C}$. When this happens, we continue the planning *on the respective face of* $\mathscr{C}$. This can be seen as initiating a new search tree, in $n-1$-dimensional space, rooted at the intersection point of the previous tree and the face. When another robot reaches its goal position, we continue the same process, i.e. planning in the $n-2$-dimensional space and so on.

Virtual Box: If one samples new configurations within the CD $\mathscr{C}$, one can never reach the faces of $\mathscr{C}$ since the faces have measure zero relative to $\mathscr{C}$. Therefore, we propose to sample new configurations within a *larger* n-orthotope $\overline{\mathscr{C}} := [0, \overline{s_1}] \times \ldots \times [0, \overline{s_n}]$, where $\overline{s_i} \geq s_i^{\text{goal}}, i \in \{1, 2, \ldots, n\}$. We call $\overline{\mathscr{C}}$ the *virtual box*.

When a sampled configuration $\boldsymbol{q}_{\text{rand}}$ is in $\overline{\mathscr{C}} \setminus \mathscr{C}$, we first find $\boldsymbol{q}_{\text{near}}$ as the tree vertex that is closest to $\boldsymbol{q}_{\text{rand}}$ as previously. Next, when extending the tree from $\boldsymbol{q}_{\text{near}}$ towards $\boldsymbol{q}_{\text{rand}}$, we stop at $\boldsymbol{q}_{\text{cropped}}$, defined as the intersection of the segment $(\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{rand}})$ [2] with the boundary of $\mathscr{C}$, see Fig. 3.

The size of the virtual box, i.e. the values $\overline{s_i}$, also affect the performance of the planner. The greater the values $\overline{s_i}$, the more the tree is attracted towards the faces of $\mathscr{C}$ and the relative value $\overline{s_i}/s_i^{\text{goal}}$ determine the attractiveness of the face i. These attractiveness ratio can also be seen as priority given to robots as the greater ratio, the more likely the robot will reach its goal first.

In our implementation, we chose $\overline{s_1} = \overline{s_2} = \ldots = \overline{s_n} = \gamma \max_i(s_i^{\text{goal}})$, where $\gamma \geq 1$ is a design parameter. Choosing an n-orthotope with equal side lengths (i.e. a cuboid) as the virtual box favors the robots with shorter path lengths since the attractiveness ratio increases with shorter path lengths. Thus,

[2]Note that for simplicity of the discussion, here we ignore the maximum extension distance ε. This maximum extension distance must also be taken into account in the actual implementation.

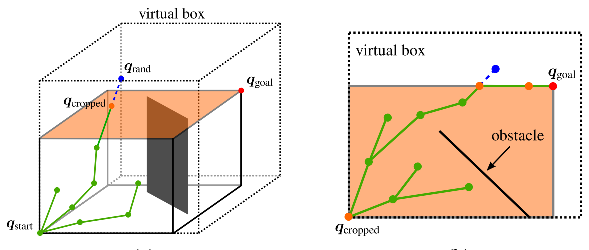


Fig. 3: Departure management. (a) At iteration k, a newly sampled configuration $\boldsymbol{q}_{\text{rand}}$ falls outside $\mathscr{C}$. The interpolated path is thresholed at $\boldsymbol{q}_{\text{cropped}}$, which lies on a face of $\mathscr{C}$ (shaded in orange). (b) From iteration $k+1$ on, the algorithm continues planning *on the face* until the search tree reaches $\boldsymbol{q}_{\text{goal}}$.

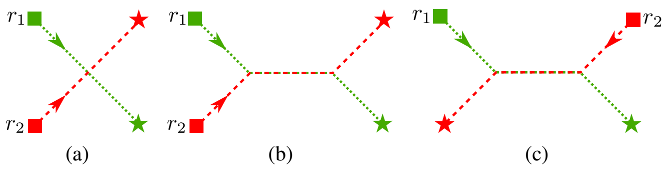


Fig. 4: Three types of elementary conflicts: (a) Junction (type-J); (b) Shared (type-S); (c) Deadlock (type-D).

the algorithm will tend to make those robots reach their goals first and "clear up the floor" more quickly, simplifying thereby the motion planning. Regarding the value γ, we found through extensive simulations that $\gamma = 1.2$ yields the best result.

B. Conflict Management

An *elementary conflict* happens when two robots, traversing their own assigned paths, have potential to collide with each other. This, in turn, happens when their paths *intersect*, either at a point or a long a segment. There are three types of elementary conflicts:

Junction (J): two robot paths cross each other at a point;
Shared (S): two robot paths share a path segment where both robots travel in the same direction; and
Deadlock (D): two robot paths share a path segment where the two robots travel in opposite directions.

The scenarios of elementary conflicts are depicted in Fig. 4.

A sampling-based planner, equipped with a collision checker[3], is able to explore the configuration space without the need to explicitly computing the obstacle set, which is computationally expensive. However, we found out that:

- The presence of narrow passages—regions in $\mathscr{C}$ with relatively small volumes that when removed, increase the number of connected component in $\mathscr{C}$—greatly affects the performance of the planner. While it is possible to use some heuristics to alleviate the problems arisen from narrow passages (see, e.g. [12]), they often require some assumptions on the obstacle set $\mathscr{O}$. Therefore, it is better to have at least partial knowledge of $\mathscr{O}$ in order to devise policies to overcome narrow passages.
- The obstacles in $\mathscr{C}$ that are induced by elementary conflicts can be closely described by simple geometric shapes and do not require extensive computational resource to compute and store them.

[3]A collision checker is a function that takes a configuration as its input and return a Boolean value indicating if the input configuration is in collision.

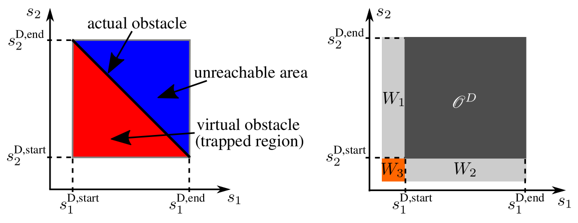


Fig. 5: Deadlock obstacle $\mathscr{O}^D$. (a) The black diagonal line represents configurations where actual collisions occur. The red region is a trapped region where there exists no collision-free and monotonically increasing path that connects a configuration inside to the goal. (b) The waiting areas W_1, W_2, and W_3. When the search tree is extended from a configuration inside a waiting area, one robot should be made temporarily inactive (i.e. waiting) until the other robot has exited the deadlock path segment.

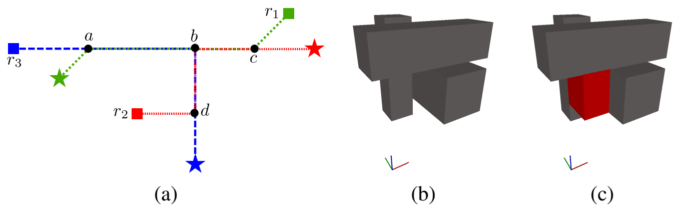


Fig. 6: Complex deadlocks. (a) The situation consists of three robots. (b) There are three elementary deadlock obstacles in $\mathscr{C}$, depicted as gray rectangular prisms (not to scale). (c) The three elementary deadlock obstacles induce a complex deadlock shown in red.

Therefore, our approach is to compute the obstacle sets induced by conflicts and to devise policies to overcome those obstacles.

1) Representing Conflicts in $\mathscr{C}$: For simplicity of discussion, we suppose that the collision radius r is very small. In practice, the radius r can be taken into account by padding the obstacle set by the value r.

Consider when there are two robots, r_1 and r_2, in the scene. The paths of the two robots are parameterized by parameters $s_1 \in [0, s_1^{\text{goal}}]$ and $s_2 \in [0, s_2^{\text{goal}}]$, respectively.

Junction: The paths intersect at $(s_1, s_2) = (s_1^J, s_2^J)$. The obstacle set $\mathscr{O}^J$ is described as $\mathscr{O}^J = \{(s_1^J, s_2^J)\}$.

Shared: Suppose the robot r_1 is in the shared segment when $s_1 \in I_1^S := [s_1^{\text{S,start}}, s_1^{\text{S,end}}]$ and the robot r_2 is in the shared segment when $s_2 \in I_2^S := [s_2^{\text{S,start}}, s_2^{\text{S,end}}]$. The obstacle set is described as $\mathscr{O}^S = \{(s_1, s_2) \,|\, s_1 \in I_1^S, s_2 \in I_2^S, s_1 = s_2 + (s_1^{\text{S,start}} - s_2^{\text{S,start}})\}$[4], which is a straight line connecting the points $(s_1^{\text{S,start}}, s_2^{\text{S,start}})$ and $(s_1^{\text{S,end}}, s_2^{\text{S,end}})$ in $\mathscr{C}$.

Deadlock: Suppose the robot r_1 is in the shared segment when $s_1 \in I_1^D := [s_1^{\text{D,start}}, s_1^{\text{D,end}}]$ and the robot r_2 is in the shared segment when $s_2 \in I_2^D := [s_2^{\text{D,start}}, s_2^{\text{D,end}}]$. The obstacle set is described as $\mathscr{O}^{D'} = \{(s_1, s_2) \,|\, s_1 \in I_1^D, s_2 \in I_2^D, s_1 + s_2 = (s_1^{\text{D,start}} + s_2^{\text{D,start}})\}$, which is a straight line connecting $(s_1^{\text{S,start}}, s_2^{\text{S,end}})$ and $(s_1^{\text{S,end}}, s_2^{\text{S,start}})$ in $\mathscr{C}$.

[4]Note that since all the paths are parameterized by their own path lengths, the intervals I_1 and I_2 have the same length. This implies $s_1^{\text{S,start}} - s_1^{\text{S,start}}$ and $s_1^{\text{S,end}} - s_1^{\text{S,end}}$, and so on.

Note, however, that the deadlock obstacle $\mathscr{O}^{D'}$ also creates a *virtual obstacle* or a *trapped region* (the red region in Fig. 5). For any configuration (s_1, s_2), although they are collision-free, there exists no monotonically increasing path that connects (s_1, s_2) to $(s_1^{\text{goal}}, s_2^{\text{goal}})$. Therefore, the search tree should not extend itself to any configuration inside any trapped region. Also, the area *above* $\mathscr{O}^{D'}$ (the blue region in Fig. 5) is not reachable due to the "no-backward-movement" constraint. Therefore, we propose to describe a deadlock obstacle as $\mathscr{O}^D = I_1^D \times I_2^D$, which is the Cartesian product of the two intervals.

When there are n robots in the scene, the obstacles are simply protrusions of the previously discussed obstacle sets into the new dimensions. For example, from the scenario in Fig. 2(a), the deadlock obstacle is a rectangular prism $I_1^D \times I_2^D \times [0, s_3^{\text{goal}}]$.

2) Resolving Conflicts: Here we only discuss a policy to overcome deadlock obstacles since they have the most volume compared to obstacles of other types hence highest potential to create narrow passages.

Since one robot can be inside a deadlock path segment at a time, when one robot, say r_1, has already entered the deadlock, the other, say r_2, needs to *wait* until r_1 robot has come out. Also, r_2 should be waiting only when it is near the entrance of the deadlock segment. Geometrically, this policy leads us to define waiting areas W_1, W_2, and W_3 in $\mathscr{C}$, as depicted in Fig. 5(b). When the search tree is extended from a configuration inside a waiting area, one robot should be made temporarily inactive to prevent future collision. If the tree is extended from $\boldsymbol{q}_{\text{near}} \in W_1$, the robot r_1 should be made waiting, i.e. the tree extension will go *vertically*. In case of W_2, the robot r_2 should be waiting, while in case of W_3, the user has freedom to choose the waiting robot. Note that the dimension of the waiting area can be adjusted to suit different applications.

3) Complex Deadlocks: Apart from elementary deadlock obstacles, as discussed previously, there exists *complex deadlocks*, i.e. ones that involve more than two robots. Note that this complex deadlock is *not* equivalent to multiple elementary deadlocks since it is not a subset of any combination of elementary obstacles.

To see this, consider a situation depicted in Fig. 6(a) in which there are three robots. There is a deadlock between every possible pair of robots. Each of the conflicts induces a deadlock obstacle in $\mathscr{C}$ that is a 3-orthotope. However, in this case, the arrangement of the three 3-orthotopes is such that it induces a new trapped region not inside any existing elementary deadlock obstacles (red region in Fig. 6(c)). We call this trapped region a *complex deadlock* as it is induced by multiple elementary obstacles. To prevent the search tree from being stuck in a complex deadlock, we present an algorithm to compute a complex deadlock obstacle induced by a cluster of elementary deadlock obstacles.

Consider an exemplary 2D coordination diagram with two deadlock obstacles, $\mathscr{O}_1^D$ and $\mathscr{O}_2^D$, connected at their corners, as depicted in Fig. 7(a). In this case, there exists a complex deadlock $\mathscr{O}_3^D$, shaded in red. To compute $\mathscr{O}_3^D$, we need to compute *shadows* produced by each elementary obstacle. Then a complex deadlock will be an intersection of shadows. More precisely, suppose there is a light source placed at $+\infty$ of s_1-axis of $\mathscr{C}$. Then $\mathscr{O}_1^D$ will produce a shadow S_1 and $\mathscr{O}_2^D$ will produce a shadow S_3 (see Fig. 7). Configurations in these shadows can be seen as being *blocked* in s_1-direction. Suppose there is another light source placed at $+\infty$ of s_2-axis then we compute shadows S_2 and S_4. Similarly, configurations in S_2 or S_4 can be seen as being blocked in s_2-direction. Finally, the complex deadlock is obtained as an intersection of shadows from different direction. This is because a deadlock is, by definition, a region that is blocked in every direction. In this example, $\mathscr{O}_3^D = S_1 \cap S_4$.

Note that the proposed procedure of computing complex deadlocks can be easily generalized to handle complex deadlocks or any dimension. For example, consider an obstacle described by a Cartesian product $[s_1', s_1''] \times [s_2', s_2''] \times \ldots \times [s_k'.s_k'']$. The shadow of this obstacle in s_1-direction is simply described as $[0, s_1'] \times [s_2', s_2''] \times \ldots \times [s_k'.s_k'']$.

V. Simulations

A. Planning Pipeline

We propose the following pipeline to address multi-robot path planning problems:

Step 1 Plan robot paths in the physical map independently using, e.g. a graph search algorithm;

Step 2 Find a collision-free path in the coordination diagram using the set of algorithms developed previously;

Step 3 Post-process the path.

In the post-processing step, one can first use shortcutting algorithms [13], [14] to smoothen the path in the CD. Then one can *time-parameterize* the physical paths, i.e. impose velocity and acceleration constraints, by using algorithms such as Time-Optimal Path Parameterization (TOPP) [15][5]. If the commands are to be transmitted by Air Traffic Controllers, one can also impose the switch times (the time duration between two acceleration switches) to be always larger than some predefined minimum value [14].

B. Simulation Results

Here we compare performance of four variants of the RRT planner [5] in coordination scenarios on the CDG airport (see Fig. 1) involving 2 to 25 aircraft. A video of a coordination solution of a problem with 25 aircraft can be found at `https://youtu.be/hfJeUKpeeD0`. All the simulations were run on a 2.2GHz laptop running Ubuntu.

RRT Variants: The four planners are as follows:

1) Vanilla RRT: This planner is without handling of departure events (virtual box) and conflict management. After each successful extension of the search tree, the planner will attempt to connect the newly added tree vertex to the goal configuration via a straight path with a probability p_{bias}.

[5]Note that the time-parameterizability also depends on the geometry of the path in the CD (the readers are referred to [15] for more details).

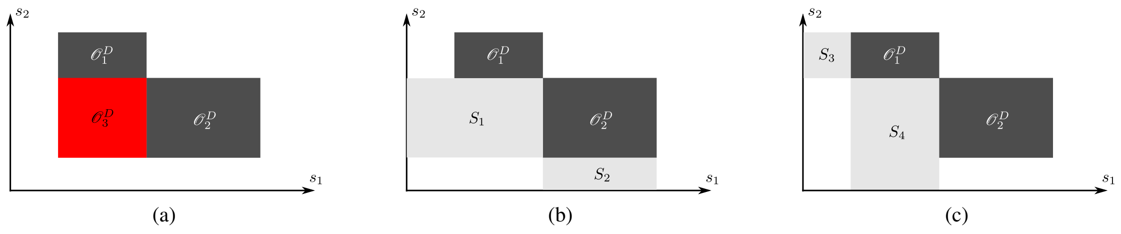


Fig. 7: Complex deadlock computation. (a) There are two elementary obstacles $\mathcal{O}_1^D$ and $\mathcal{O}_1^D$, which are connected at their corners. These elementary obstacles induce a complex deadlock $\mathcal{O}_3^D$, shaded in red. To compute $\mathcal{O}_3^D$, we compute shadows of $\mathcal{O}_1^D$ (see (b)) and shadows of $\mathcal{O}_1^D$ (see (c)). Then the complex deadlock is the non-empty intersection between shadows of different directions. In this case, $\mathcal{O}_3^D = S_1 \cap S_4$.

2) RRT-DM: This is an RRT planner with only the **D**eparture **M**anagement feature, i.e. the virtual box heuristic.
3) RRT-CM: This is an RRT planner with only the **C**onflict **M**anagement feature. Since the virtual box heuristic is not used, there is also a connection attempt after each successful tree extension as in the vanilla RRT.
4) RRT-DCM: This is an RRT planner with both the departure management and conflict management features.

In the simulations, all variants used the same maximum extension distance (ε). The probability p_{bias} (for Vanilla RRT and RRT-CM) was set to 0.2.

Problem Settings: Let $2 \leq n \leq 25$ be the number of robots in the scene. For each n, we first randomly sampled n pairs of start and goal positions. Then for each planner, we ran simulations 50 times on each of the 24 sampled problem instances. In each run, the time limit was set to 60 s.

Results are reported in Fig. 8. While RRT-DCM found a solution in every run, the success rates of the other three planners decrease rapidly with n. At large n's, those planners only occasionally get *lucky* enough to find solutions. From this sufficiently clear trend, we did not run simulations for Vanilla RRT, RRT-DM, and RRT-CM for $n > 15$.

Interestingly, RRT-DM performed better than Vanilla RRT and RRT-CM. This is because in a multi-robot path coordination scenario, allowing robots to reach their goals at different times greatly increases the solution space, hence the high success rate. However, since a new search tree is initiated when a robot reaches its goal, the lack of conflict management feature can result in the new tree being trapped completely in a deadlock. Similarly, although the conflict management feature is useful, not explicitly allowing robots to reach their goals at different times is still too restrictive that although the search tree did not get trapped, the planner required too much time to find a solution.

One can see that while the presented two features—departure handling and conflict management—are not much beneficial being used separately, they are complementary. Therefore, integrating both of them into a planner significantly improved the performance as illustrated in simulations.

Note also that although previous works such as [16] presented simulations with 240 aircraft, the number of aircraft was cumulative across an interval of three hours. All snapshots presented in [16] showed less than 20 aircraft in the airport. By contrast, the simulations presented here were such that all aircraft were presented in the scene.

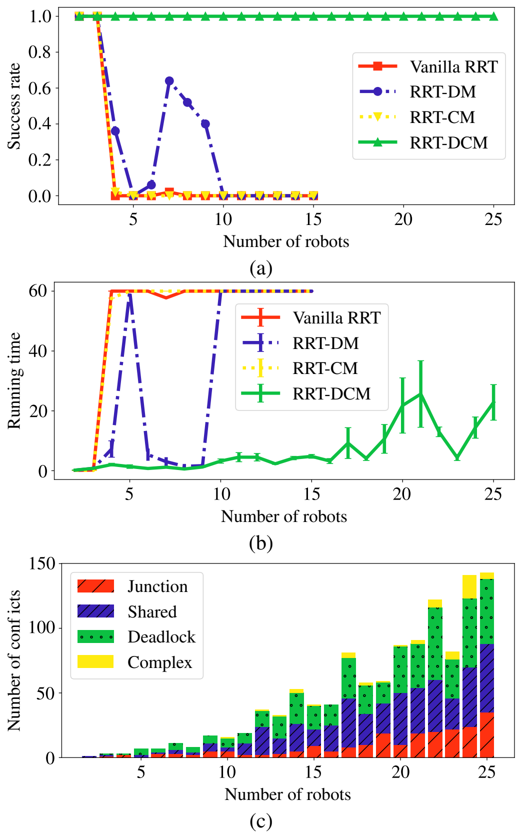


Fig. 8: Experimental results. (a) The success rates of all planners as a function of the number of involving aircraft. (b) The average running time over 50 runs of all planners as a function of the number of aircraft. The vertical bars indicate the respective standard deviation. In case a planner failed in all 50 runs, the running time of that problem instance is shown as the time limit, 60 s. (c) The number of conflicts of each type in the sampled problem instances used in the simulations.

VI. Discussion and Conclusion

A. Discussion

Arrival Management: This feature cannot be directly integrated into the presented approach since the planning in the coordination diagram does *not* include timing. The planned path only encodes information of *relative positions* of robots/aircraft. It is thus not possible to include new aircraft into the planner while in the middle of the planning process. However, the simulation results presented previously suggest that it can be possible to use the approach "plan-execute-replan". That is, it is feasible to assume that a coordinated path is planned and the execution starts before a new aircraft arrives. Thus, when a new aircraft arrives, one can *replan* the coordinated path by using the current configuration as the root of the new search tree.

Complex Deadlocks: The existence of complex deadlocks has been already been discussed in the literature (see, e.g. [1]). Similar ideas were also explored in literature on concurrency control of databases [17], [18]. However, previous works only discussed these deadlocks in 2D cases. Furthermore, a complex deadlock was also thought to be the *SW-hull* [19] of other existing obstacles [1]. Upon further investigation, we found out that the complex deadlocks are equivalent to SW-hulls only in two-dimensional cases. To our knowledge, this paper is the first to develop an algorithm to explicitly address high-dimensional complex deadlocks.

The presented approach, however, comes with high combinatorial complexity. To compute a complex deadlock induced by a cluster of m elementary deadlocks, we compute exactly two shadows produced by each obstacle. In the worst case, we need to examine all possible cases 2^m of shadow intersection (since in each case, we can select one out of two shadows). Therefore, the worst case complexity is $O(2^m)$. It is possible to reduce the computation time by using, for example, a graph search algorithm that takes into account path overlapping, hence reducing m. However, to derive a more efficient complex deadlock detection algorithm, we need to gain more understanding of how complex deadlocks are formed, their mathematical properties, etc.

Completeness: The RRT planner itself is probabilistically complete [5], meaning that if there exists a solution, it will be found with probability one as the number of iterations increases. However, the planner presented in this paper *may* not be probabilistically complete. One possible factor can be the current mechanism of departure management which reduces the dimension of the coordination diagram immediately after a robot has reached its goal. This mechanism implicitly assumes that the configuration $\boldsymbol{q}_{\text{cropped}}$ is part of some solutions, which we currently have no guarantee. Further study should also focus on this completeness issue.

B. Conclusion

In this paper, we have developed a set of algorithms to deal with departure events and conflicts (mainly caused by the "no-backward-movement" constraint) in the classical multi-robot path coordination problem. The virtual box heuristic, when integrated into a planner, allows robots to reach their goals at different times while the conflict management feature helps prevent the search tree from being trapped in a deadlock situation. Integrating the presented two features in a multi-robot coordination planner greatly improves the performance, as illustrated in simulations. Future research directions include, but not limited to, 1) further understanding of deadlock formation and how to reduce complexity of complex deadlock computation; and 2) analyzing the completeness property of the planner.

Acknowledgment

This work was partially supported by grant ATMRI:2014-R6-PHAM awarded by NTU and the Civil Aviation Authority of Singapore. We thank our colleagues from ENAC, Railane Benhacene and Mathieu Cousy, for directing us towards the aircraft taxiing problem as well as for providing the data for Charles de Gaulle airport.

References

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

[2] P. Švestka and M. H. Overmars, "Coordinated path planning for multiple robots," *Robotics and Autonomous Systems*, vol. 23, no. 3, pp. 125–152, April 1998.

[3] T. Siméon, S. Leroy, and J.-P. Laumond, "Path coordination for multiple mobile robots: a resolution-complete algorithm," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 1, pp. 42–49, 2002.

[4] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[5] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[6] Á. G. Marín, "Airport management: Taxi planning," in *Annals of Operations Research*, vol. 143, no. 1, 2006, pp. 191–202.

[7] ——, "Airport taxi planning: Lagrangian decomposition," *Journal of Advanced Transportation*, vol. 47, no. 4, pp. 461–474, 2013.

[8] M. J. S. J. W. Smeltink, "An optimisation model for airport taxi scheduling," 2004.

[9] J. Peng and S. Akella, "Coordinating multiple robots with kinodynamic constraints along specified paths," *The International Journal of Robotics Research*, vol. 24, no. 4, pp. 295–310, 2005.

[10] K. Solovey, O. Salzman, and D. Halperin, "Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning," in *Springer Tracts in Advanced Robotics*, vol. 107, 2015, pp. 591–607.

[11] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[12] Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, and J. H. Reif, "Narrow passage sampling for probabilistic roadmap planning," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1105–1115, 2005.

[13] R. Geraerts and M. H. Overmars, "Creating high-quality paths for motion planning," *The International Journal of Robotics Research*, vol. 26, no. 8, pp. 845–863, 2007.

[14] P. Lertkultanon and Q.-C. Pham, "Time-optimal parabolic interpolation with velocity, acceleration, and minimum-switch-time constraints," *Advanced Robotics*, vol. 30, no. 17–18, pp. 1095–1110, 2016.

[15] Q.-C. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *IEEE Transactions on Robotics*, vol. 30, pp. 1533–1540, 2014.

[16] G. L. Clare and A. G. Richards, "Optimization of taxiway routing and runway scheduling," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1000–1013, 2011.

[17] M. Yannakakis, C. H. Papadimitriou, and H. T. Kung, "Locking policies: Safety and freedom from deadlock," in *Foundations of Computer Science, 1979., 20th Annual Symposium on*, 1979, pp. 286–297.

[18] W. Lipski and C. H. Papadimitriou, "A fast algorithm for testing for safety and detecting deadlocks in locked transaction systems," *Journal of Algorithms*, vol. 2, no. 3, pp. 211–226, 1981.

[19] F. P. Preparata and M. Shamos, *Computational Geometry*. Springer-Verlag New York, 1985.