Departure, arrival, and deadlock management in multi-robot path coordination

Yang Jingyi, Pham Tien Hung, Puttichai Lertkultanon, and Quang-Cuong Pham

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 and arrival events, which remove robots from or add robots to the coordination diagram at arbitrary time instants. The second feature is the “no backward movements” constraint: the robots can only move forward on their assigned paths. We propose a set of algorithms to efficiently take into account these features and validate these algorithms on a realistic model of Charles de Gaulle airport.

I. INTRODUCTION

Consider \( n \) robots initially placed at positions \( P_1^{\text{start}}, \ldots, P_n^{\text{start}} \) and whose tasks are to reach goal positions \( P_1^{\text{goal}}, \ldots, P_n^{\text{goal}} \) respectively, in a static environment. Suppose furthermore that their paths from \( P_i^{\text{start}} \) to \( P_i^{\text{goal}} \) \( (i \in [1, n]) \) are fixed and do not involve collisions with the environment. Finding the coordinated motions of the robots to move along their paths while avoiding mutual collisions is a classical problem in 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, 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 step (i) fixed. In some other applications, the robots have to move along predefined “tracks”, e.g. driver-less cars on city streets or aircraft automatic taxiing on airport tracks. Here, the constraint 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 considering 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 airport in Paris). Black arrows indicate one-way tracks. B: A simplified model of the airport. Gray arrows indicate track directions. Boxes and stars indicate respectively start and goal positions. Each dashed line represents the shortest path between a pair of start and goal positions of the same color. Note that there is a large number of shared track segments and potential deadlocks, which yields a hard path coordination problem. A video of the coordination solution computed by our algorithm can be found at https://youtu.be/pJARueJ-sog.

movements” 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.

Fig. 1: A: Actual map of a part of Charles de Gaulle (CDG) airport in Paris. Black arrows indicate one-way tracks. B: A simplified model of the airport. Gray arrows indicate track directions. Boxes and stars indicate respectively start and goal positions. Each dashed line represents the shortest path between a pair of start and goal positions of the same color. Note that there is a large number of shared track segments and potential deadlocks, which yields a hard path coordination problem. A video of the coordination solution computed by our algorithm can be found at https://youtu.be/pJARueJ-sog.

The remainder 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, arrival and deadlocks, which are specifically caused by the “no backward movements” 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 one single step.

A. Non-decoupled methods

Applying classical robot motion planning methods, such as PRM [4] or 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 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 [2], 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 [3], 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 considering the features mentioned previously, namely departure, arrival and deadlock management. As we shall demonstrate, taking these features into account significantly improves computation time with respect to the vanilla RRT. Note that the development presented here can be easily combined with existing decoupled planning heuristics (e.g. roadmap coordination [3]).

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.

III. BACKGROUND: MULTI-ROBOT PATH COORDINATION WITHOUT BACKWARD MOVEMENTS

A. Coordination diagram

Consider \( n \) robots traveling on \( n \) given paths \((P_1, \ldots, P_n)\), with lengths \((l_1, \ldots, l_n)\). The positions of the robots on their respective paths can be given by an \( n \)-tuple \((q_1, \ldots, q_n)\), where \( q_i \) \((0 \leq q_i \leq l_i)\) denotes the path length traveled by robot \( i \) from the origin of its path \( P_i \). Graphically, one can thus represent this \( n \)-tuple as a point in an \( n \)-orthotope \( C := [0, l_1] \times \cdots \times [0, l_n] \), also called the coordination diagram [1], [3], [2].

As two paths \( P_i \) and \( P_j \) may intersect or even share some track segments, robots \( i \) and \( j \) may collide when traveling
along their paths. One can thus define the obstacle set in the
coordinate diagram as
\[
O := \{(q_1, \ldots, q_n) \in C : \exists i, j, \|p_i(q_i) - p_j(q_j)\| < r\},
\]
where \(p_i(q_i)\) is the 2D physical position of robot \(i\) when it
has traveled the distance \(q_i\) on its path and \(r\) is the collision
radius.

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) reach the end of their paths;
(iii) never collide with one another – can now be cast
as a classical motion planning problem: find a collision-
free continuous path between \((0, \ldots, 0)\) and \((l_1, \ldots, l_n)\)
in the coordination diagram. Fig. 2 shows an example of
coordination diagram constructed for three robot paths.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig2.png}
\caption{A: Three robot paths in the physical map. 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 gray. Note that
for simplicity of the construction, the collision radius was supposed to be
very small – otherwise, the gray surface representing the obstacle set would
be thicker. The search tree is depicted in green and the solution path is
highlighted in red.}
\end{figure}

The classical motion planning problem can be solved by
any existing geometric motion planning algorithms [11].
Here, we use RRT [5]. Briefly, RRT iteratively grows a
tree rooted at the start configuration \(q_{\text{start}} := (0, \ldots, 0)\).
At each iteration, a random configuration \(q_{\text{rand}}\) is sampled
within the free space \(C \setminus O\). Next, one determines, from
amongst the existing tree vertices, the vertex \(q_{\text{near}}\) that is
the nearest neighbor to \(q_{\text{rand}}\) according to some distance
function \(\text{dist}(q_{\text{near}}, q_{\text{rand}})\). Finally, one tries extending the
tree from \(q_{\text{near}}\) towards \(q_{\text{rand}}\), creating thereby a new tree vertex \(q_{\text{new}}\).
The algorithm stops when the goal configuration \(q_{\text{goal}} := (l_1, \ldots, l_n)\)
can be connected to the tree. For more
details and variations, the reader is referred to [5].

Collision checks: When extending from \(q_{\text{near}}\) towards \(q_{\text{rand}}\),
one needs to check whether the segment \(q_{\text{near}} q_{\text{rand}}\) is
collision-free. For this, one checks whether all discretized
configurations \(q\) along the segment \(q_{\text{near}} q_{\text{rand}}\) are collision-
free.

Consider a configuration \(q := (q_1, \ldots, q_n)\), to check
whether this configuration is collision free, one puts all the
robots \(q_i, i \in [1, n]\), to the physical positions defined by
the \(q_i\) and checks for all pairs \((i, j)\) whether robot \(i\) and
\(j\) are mutually in collision, according to [11]. Note that the
obstacle set is therefore never explicitly constructed in the
coordination diagram.

B. Motion planning without backward movements

Consider a straight path from \(q := (q_1, \ldots, q_n)\) to \(q' :=
(q'_1, \ldots, q'_n)\) in the coordination diagram. The “no backward
movement” constraint can be rendered as \(q \preceq q'\) where
\[
q \preceq q' \text{ if and only if } \forall i \in [1, \ldots, n], q_i \leq q'_i.
\]
To enforce this constraint in RRT, one should attempt exten-
sion from \(q_{\text{near}}\) to \(q_{\text{rand}}\) only when \(q_{\text{near}} \preceq q_{\text{rand}}\). A simple
way to implement this filter is to modify the distance function
(used in the nearest neighbor search) as
\[
\text{dist}(q \rightarrow q') := \begin{cases}
\|q' - q\| & \text{if } q \preceq q' \\
+\infty & \text{if } q \npreceq q'
\end{cases},
\]
such that any \(q \preceq q_{\text{rand}}\) will never be selected as the nearest
neighbor of \(q_{\text{rand}}\).

IV. DEPARTURE, ARRIVAL, AND DEADLOCK
MANAGEMENT

A. Departure management

We say that a robot departs from the diagram 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 the 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 coordination diagram, a departure corresponds to the
search tree reaching a configuration \(q\) on a \(n - 1\)-dimension
face of the \(n\)-orthotope, for instance \(q = (q_1, \ldots, q_{n-1}, 1)\).
When this happens, we project the coordination diagram onto
the \(n - 1\)-dimension face and start a new tree rooted at
\((0, \ldots, 0, 1)\). See Fig. 3.

Collision checks: Since the obstacles are not constructed
explicitly, the projection operation is transparent in terms of
collision checks: in the \(n - 1\)-dimension face, one simply
does not check for possible collisions involving robot \(n\).

Completeness: RRT has been proved to be probabilistically
complete, meaning that, if there exists a solution, it will
be found with probability 1. One possible concern is that
completeness is lost through the projection. This is addressed
in Section IV-C where we discuss the deadlock
issue.

Virtual box: If one samples within the \(n\)-orthotope \(C\), one
can never reach the faces, since the faces have measure 0.
Therefore, we propose to sample within a larger \(n\)-orthotope
\footnote{In case \(d(q_{\text{near}} \rightarrow q_{\text{rand}}) < \epsilon, \) where \(\epsilon\) is the maximum extension
distance, one considers instead the segment \(q_{\text{near}} q_{\text{close}}\), where \(q_{\text{close}}\) is
the configuration on the segment \(q_{\text{near}} q_{\text{rand}}\) at distance \(\epsilon\) from \(q_{\text{near}}\).

\footnote{Alternatively, one can re-initialize the paths of robots 1 to \(n - 1\) so
that these paths begin at the current position and start a new tree rooted at
\((0, \ldots, 0)\).}
When a configuration \( q_{\text{rand}} \) is sampled in \( \bar{C} \), we first find \( q_{\text{near}} \) as the vertex of the tree that is the closest to \( q_{\text{rand}} \) as previously. Next, when extending from \( q_{\text{near}} \) towards \( q_{\text{rand}} \), we stop at \( q_{\text{proj}} \), defined as the intersection of the segment \( (q_{\text{near}}, q_{\text{rand}}) \) with the boundary of \( C \), see Fig. 3.

The values of the \( l_i \)'s can affect the performance of the virtual box algorithm: the larger the \( l_i \)'s, the more the tree is attracted towards the faces, and the relative values of the \( l_i \)'s determine which faces are more attractive.

In the implementation, we chose \( l_1 = \cdots = l_n = c \max(l_1, \ldots, l_n) \), where \( c > 1 \) is a design parameter. Choosing a \( n \)-orthotope with equal length edges (a cuboid) actually favors the robots with the shortest path length, since the ratio \( l_i > l_i \) will be the largest when \( l_i \) is the smallest. Thus, the algorithm will tend to make those robots terminate first and “clear up the floor” more quickly, simplifying thereby the motion planning. Next, regarding the value of \( c \), we found through extensive simulations that \( c = 1.2 \) yields the best result.

B. Arrival management

When a new robot is added to the motion planning problem, we say that it arrives into the coordination diagram, also in line with the aircraft taxiing context, where aircraft arriving (landing) at the airport or driving out from the gate.

Here, we extend the coordination diagram by adding a new dimension \( n + 1 \). We then start a new tree rooted at \( (q_1, \ldots, q_n, 0) \). As in the departure case, the dimension increase does not involve any difficulty with regards to collision checks. Contrary to the departure case, there is no concern here about completeness or deadlocks.

C. Deadlock management

Consider again the situation depicted in Fig. 2A. As remarked previously, robots 1 and 2 share a section of their paths. To make things worse, the two robots travel in opposite directions on their common path segment, which would potentially create a deadlock situation. Assume that robot 1 is at position \( a \) and robot 2 is at position \( b \). Since the robots are not allowed to move backward, this situation will result in a planning failure.

Coming back to the remark on completeness issue evoked in Section IV-A, consider the configuration where robots 1, 2 are at \( a, b \) respectively and robot 3 is at its goal position. This configuration is collision-free, but if one projects the coordination diagram onto the face corresponding to robots 1 and 2 and start a new tree at \((a, b)\), then no solution can be found. This example shows that the departure management proposed in Section IV-A can incur a loss in completeness if deadlocks are not properly addressed. Moreover, even when a solution is eventually found, deadlocks considerably slow down the search.

1) Representing deadlocks: Deadlocks occur whenever a path segment is shared by two robots traveling in opposite directions. We propose to record all possible deadlocks in a pre-processing step (before running the search algorithm) as in Algorithm 1.

### Algorithm 1: Find all deadlocks

**Input**: \( n \) physical paths

**Output**: The list of all deadlocks

1. for \( i \in [1, \ldots, n] \) do
2. for \( p \) portion of \( P_i \) do
3. for \( j \in [i + 1, \ldots, n] \) do
4. if \( p \) is a portion of \( P_j \) and traveled by robot \( j \) in the opposite direction then
5. Deadlocklist.append((\( i, j, p \));

This algorithm has a complexity \( O(n^2 M^2) \), where \( M \) is the maximum number of track segments per robot path. In practice, this pre-processing step takes a few milliseconds and does not significantly affect the computation time of the overall algorithm.

Next, during the search itself, given a configuration \( q \), one can check whether \( q \) gives rise to a deadlock by cycling through the items of Deadlocklist and check whether any of them is triggered by the particular values of the \( q_i \)'s. In practice, as the number of deadlocks is small with respect to the total number of track segments (see Fig. 5A), this check is very fast, as compared, e.g., to collision checks.

2) Overtaking deadlocks: To avoid losing completeness, one can simply check whether a configuration is deadlocked before performing any projection. Going further, it turns out that addressing deadlocks preemptively can significantly improve search time, as it avoids creating useless (and potentially harmful) vertices in the deadlock regions.

Accordingly, we suggest to overtake deadlocks as follows: when a configuration \( q_{\text{rand}} \) is sampled near an edge \( e \) of a
deadlock region, one creates a new vertex \( q_{\text{over}} \) that extends the tree from \( q_{\text{rand}} \) and overtakes the deadlock region along \( e \), see Fig. 4. In the physical space, this operation implies that one of the robots involved in the deadlock moves to the end of the deadlock track segment, while all the other robots remain stationary.

One possible drawback of this strategy is that it may slow down execution time since all robots except one is stationary during the deadlock overtaking phase. This, however, can be addressed at the post-processing stage, see Section V-A.

V. IMPLEMENTATION AND SIMULATIONS

A. Pipeline

So far, we have addressed the problem of path coordination. Here, we integrate this step within a full multi-robot planner, following the pipeline below:

1) Plan robot paths independently in the physical map using graph search (breadth first search can be used since the graph is not too big);
2) Find a collision-free path in the coordination diagram using the algorithms previously developed;
3) Improve the coordination quality by shortcutting [12], [13] in the coordination diagram;
4) Using the coordination path, time-parameterize the physical paths.

In step 1, heuristics that penalize path sharing can make the subsequent coordination problem substantially easier. However, we did not implement such heuristics here, as the main objective is to develop efficient coordination algorithms.

In step 4, one can impose velocity and acceleration constraints on the physical trajectories, using such algorithm as Time-Optimal Path Parameterization (TOPP) [14]. If the commands are to be transmitted by Air Traffic Controllers, one can impose the switch times (the time duration between two velocity/acceleration switches) to be always larger than some minimum value by using [13].

B. Simulation results

We tested our implementation on the airport model of Fig. 1B in coordination scenarios involving 2 to 10 aircraft, see a video of coordination for 10 aircraft at [https://youtu.be/pJARueJ-sog](https://youtu.be/pJARueJ-sog). For each \( n \), where \( n \) is the number of aircraft, \( n \) start positions and \( n \) goal positions were picked randomly from the 10 start and 10 goal positions in Fig. 1B, and paired randomly to obtain \( n \) simultaneous requests. Following the pipeline just presented, the \( n \) requests are first solved independently at the path level. The resulting paths are then given to the coordination algorithm. For each \( n \), we repeated this 100 times, yielding 100 coordination problems.

Note that the coordination problems considered are hard, in the sense that a significant proportion of the tracks are shared, see Fig. 1B. Fig. 5A makes this observation more precise by showing the total number of track segments that are used by at least one aircraft, the number of track segments that are shared by at least two aircraft going in the same direction, and the number of deadlocked segments. As an example, for 10 aircraft, about 37% of the tracks are shared, and in particular 7% are deadlocked.

To solve the coordination problem (step 2 of the pipeline), we used a re-start paradigm: the coordination algorithm was given 10 s to find a valid path in the coordination diagram. If it fails, it can re-start twice. Thus, after at most 30 s, the algorithm has either found one valid coordination or reports failure. The percentage of failures is given in Fig. 5B. The average computation time for trials that succeeded is given in Fig. 5C. One can see that departure management by the virtual box method and deadlock management significantly improved search time and success rate, as compared to the vanilla RRT. In particular, the low search time opens the possibility of using our algorithms in real-time airport applications.

Fig. 5D show the trajectory duration, that is, the time used by the aircraft that reaches its goal the latest, with or without deadlock management. One can see that, while overtaking deadlocks might result initially in slightly longer trajectories (since, during the overtaking phase, all robots but one are stationary), the shortcutting process makes the trajectories with deadlock management as fast as those without deadlock management.

VI. Conclusion

We have developed a set of algorithms to take into account departures, arrivals, and deadlocks (caused by the “no backward movements” constraint) in the classical path coordination problem. We have implemented these algorithms on a realistic airport taxiing example and showed that they yield significant improvements in terms of search time with respect to existing coordination algorithms. The methods developed here can be straightforwardly combined with existing heuristics, such as better graph search to minimize path sharing in the path planning phase, roadmap coordination [3], optimal motion planning, etc. Another direction of research is to integrate these algorithms with
existing airport simulation software and conduct experiments with Air Traffic Controllers to assess the suitability of this approach for real-time taxiing control.

Acknowledgment

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


Fig. 5: Simulation results. A: Average ($\pm$ std) number of track segments used, shared without deadlock, and deadlocked. B: Failure rate for different versions of the coordination algorithm. C: Average ($\pm$ std) computation time. D: Average ($\pm$ std) duration of the longest trajectory (last aircraft reaching its goal position), before and after shortcutting.