A Loosely-Coupled Approach for Multi-Robot Coordination, Motion Planning and Control

Federico Pecora, Henrik Andreasson, Masoumeh Mansouri, Vilian Petkov
Center for Applied Autonomous Sensor Systems, Örebro University
<name>.<surname>@oru.se

Abstract

Deploying fleets of autonomous robots in real-world applications requires addressing three problems: motion planning, coordination, and control. Application-specific features of the environment and robots often narrow down the possible motion planning and control methods that can be used. This paper proposes a lightweight coordination method that implements a high-level controller for a fleet of potentially heterogeneous robots. Very few assumptions are made on robot controllers, which are required only to be able to accept set point updates and to report their current state. The approach can be used with any motion planning method for computing kinematically-feasible paths. Coordination uses heuristics to update priorities while robots are in motion, and a simple model of robot dynamics to guarantee dynamic feasibility. The approach avoids a priori discretization of the environment or of robot paths, allowing robots to "follow each other" through critical sections. We validate the method formally and experimentally with different motion planners and robot controllers, in simulation and with real robots.

Introduction

Motion planning, coordination and control are all essential for deploying fleets of autonomous robots. These three problems are intrinsically dependent: robot motions must be physically realizable by controls computed by robot controllers, and must also be coordinated in order to avoid collisions and deadlocks. Although methods for addressing these problems jointly have been studied (see Related Work), real-world applications often pose further requirements that narrow down the range of methods that can be used. Different motion planning strategies are applicable in different environments and for different types of robots (Elbanhawi and Simic, 2014). Fleets may include different types of robots with fundamentally different control schemes (Chung, Fu, and Kröger, 2016), and robot controllers are often certified black boxes that ship with the robot platform and cannot be modified. In practice, there are many reasons for considering motion planning, coordination and control separately.

In this paper, we propose a lightweight, centralized coordination method that implements a high-level controller for a fleet of potentially heterogeneous robots. Goals can be posted to one or more robots while the fleet is in motion. The method makes very few assumptions on robot controllers, and is not specific to a particular motion planning method for computing kinematically (or kinodynamically) feasible paths. The method avoids the need for a priori discretization of the environment or of robot paths, and allows robots to "follow each other" through critical sections. Also, coordination can be made to account for robot dynamics via a simple forward model. This enables the use of heuristics for revising precedences of robots while the fleet is in motion. We validate the method formally and experimentally with different motion planners and robot controllers, in simulation and with real robots.

Spatial Envelopes and Critical Points

Let $\mathcal{Q}$ be a configuration space, and let $\boldsymbol{p} : [0,1] \rightarrow \mathcal{Q}$ denote a *path* for a robot in the configuration space $\mathcal{Q}$, parametrized using its arc length σ. Hence, $\boldsymbol{p}(0)$ denotes the starting configuration, and $\boldsymbol{p}(1)$ denotes the final configuration of the robot. Given a temporal profile along the path $\sigma = \sigma(t)$, we refer to $\boldsymbol{p}(\sigma)$ as a trajectory. Given any set of configurations $S \subseteq \bigcup_{\sigma\in[0,1]} \boldsymbol{p}(\sigma)$, let $\inf_{\boldsymbol{p}} S = \arg\min_{q\in S} \boldsymbol{p}^{-1}(q)$ and $\sup_{\boldsymbol{p}} S = \arg\max_{q\in S} \boldsymbol{p}^{-1}(q)$, i.e., the configurations among those in S that are reached first and last along the path $\boldsymbol{p}$, respectively. Also, let $\boldsymbol{p}^{[t',t'']} = \bigcup_{t\in[t',t'']} \boldsymbol{p}(\sigma(t))$. We use $(\cdot)_j$ to indicate that variable $(\cdot)$ is associated to robot j. When there is no ambiguity, the subscript j will be omitted.

Let $R_j(q)$ be the transformation of robot j in a configuration $q \in \mathcal{Q}$. In the example in Figure 1 (top), $R_2(q)$ is a polygon in $\mathbb{R}^2$ representing the footprint of robot 2 in pose q. Given the set of obstacles $\mathcal{O}$, let $\mathcal{Q}_j = \{q \in \mathcal{Q} \mid \exists O \in \mathcal{O} : R_j(q) \cap O \neq \emptyset\}$, and $\mathcal{Q}_j^{\text{free}} = \mathcal{Q} \setminus \mathcal{Q}_j$.

Definition 1. Given a robot j with configuration space $\mathcal{Q}_j^{\text{free}}$, a starting configuration q^s, and a goal configuration q^g, the *path planning problem* is the problem of finding a path $\boldsymbol{p}_j$ such that $\boldsymbol{p}_j(0) = q^s$, $\boldsymbol{p}_j(1) = q^g$, and $\boldsymbol{p}_j(\sigma) \in \mathcal{Q}_j^{\text{free}}, \forall\sigma \in [0,1]$, typically subject to differential constraints $f(q,\dot{q}) = 0$.

In order for a path to be executable by a robot, a suitable temporal profile needs to be computed:

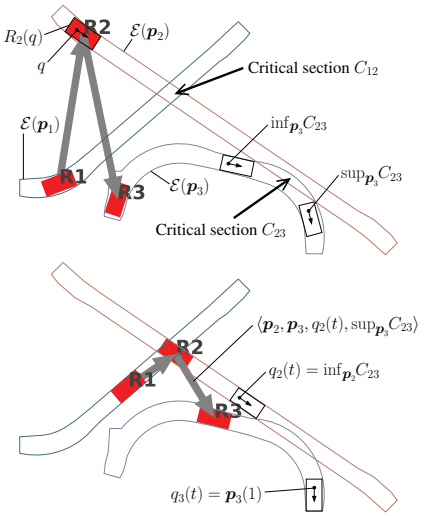

Figure 1: Three robots navigating along paths $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, and $\boldsymbol{p}_3$. Spatial envelopes and critical sections are shown above; gray arrows indicate precedence constraints; detail of the precedence constraint regulating robots 2 and 3 as they navigate through C_{23} is shown below.

Definition 2. Given a path $\boldsymbol{p}$, the *trajectory generation problem* is the problem of synthesizing an executable temporal profile $\sigma(t)$ for $\boldsymbol{p}$, typically subject to differential constraints $g(\dot{q}, \ddot{q}) = 0$.

Trajectory generation is typically done by the *robot controller*, and is achieved as part of the larger problem of synthesizing control actions for the robot. In doing so, robot control schemes typically account for robot dynamics (e.g., the robot's mass) and a variety of other constraints (e.g., on the range of control inputs). Constraints that account for robot-robot collisions are seldom part of the control problem formulation (although exceptions exist, see Related Work), as they increase the computational effort required to solve the control problem. In this paper, we are interested in preserving to the greatest extent possible the formulation of the robot control problem. As we will see, our approach is to communicate to robot controllers the simplest possible further constraints necessary to avoid collisions.

Definition 3. The *spatial envelope* $\mathcal{E}(\boldsymbol{p})$ of a path $\boldsymbol{p}$ is the set of robot transformations reached along the path, that is, $\mathcal{E}(\boldsymbol{p}) = \bigcup_{\sigma \in [0,1]} R(\boldsymbol{p}(\sigma))$.

Definition 4. Given two robots i and j, we say that paths $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ *interfere* iff $\mathcal{E}(\boldsymbol{p}_i) \cap \mathcal{E}(\boldsymbol{p}_j) \neq \emptyset$.

Definition 5. Given two paths $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, let $\mathcal{S} = \{q \in \mathcal{Q} \mid R_i(q) \cap \mathcal{E}(\boldsymbol{p}_j) \neq \emptyset \vee R_j(q) \cap \mathcal{E}(\boldsymbol{p}_i) \neq \emptyset\}$, and let $\mathcal{C}_{ij}$ be the decomposition of $\mathcal{S}$ into its largest contiguous subsets. Each set of configurations $C_{ij} \in \mathcal{C}_{ij}$ is called a *critical section*.

It follows from Definition 5 that $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ interfere if and only if $\mathcal{C}_{ij} \neq \emptyset$. Equivalently (see Definitions 3 and 4), $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ interfere iff there exist temporal profiles σ_i and σ_j such that $R_i(\boldsymbol{p}_i(\sigma_i(t))) \cap R_j(\boldsymbol{p}_j(\sigma_j(t))) \neq \emptyset$ for some time t. Note also that for any $C_{ij} \in \mathcal{C}_{ij}$ and any $\sigma \in [0,1]$, we have that $R_i(\boldsymbol{p}_i(\sigma)) \cap C_{ij} \neq \emptyset$ iff $\ell \leq \sigma \leq u$, for some $\ell, u \in [0,1]$. The configuration in which robot i begins to intersect C_{ij} is $\inf_{\boldsymbol{p}_i} C_{ij} = \boldsymbol{p}_i(\ell)$, and the configuration just before the same robot ceases to intersect C_{ij} is $\sup_{\boldsymbol{p}_i} C_{ij} = \boldsymbol{p}_i(u)$ (similarly for robot j).

Figure 1 shows an example of three spatial envelopes with two critical sections. We assume (as is the case in the example) that $\boldsymbol{p}_i^{-1}(\inf_{\boldsymbol{p}_i} C_{ij}) > 0$ and $\boldsymbol{p}_i^{-1}(\sup_{\boldsymbol{p}_i} C_{ij}) < 1$ for all robots i and critical sections C_{ij}, that is, no robot begins or terminates its motion in a critical section. The consequences of relaxing this assumption are further discussed when we consider deadlocks.

Definition 6. A *precedence constraint* is a tuple $\langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i, q_j \rangle$ expressing the following time-dependent constraint on the temporal profiles of $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$:

$$q_j \notin \boldsymbol{p}_j^{[0,t]} \Rightarrow q_i \notin \boldsymbol{p}_i^{[0,t]}. \tag{1}$$

A precedence constraint $\langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i, q_j \rangle$ can be read as follows: robot i should not navigate beyond configuration q_i along path $\boldsymbol{p}_i$ until robot j has reached configuration q_j along path $\boldsymbol{p}_j$. A precedence constraint limits the possible temporal profiles of robot i. Whether or not it does so depends on the time at which it is evaluated (t) and on the temporal profile of robot j. In Figure 1 (bottom), robot 2 is subject to a constraint regulating its access to C_{23} at time t. We use precedence constraints to regulate access and traversal of critical sections so as to avoid collisions (whereas no such regulation is necessary outside critical sections). This poses the following problem:

Definition 7. Given a set of paths $\mathcal{P}$ for an arbitrary number of robots, the *coordination problem* is the problem of synthesizing, for each pair of interfering paths $(\boldsymbol{p}_i, \boldsymbol{p}_{j \neq i}) \in \mathcal{P}^2$, constraints on the temporal profiles $\sigma_i(t)$ and $\sigma_j(t)$ such that $R_i(\boldsymbol{p}_i(\sigma_i(t))) \cap R_j(\boldsymbol{p}_j(\sigma_j(t))) = \emptyset, \forall t$.

Remark 1. Let $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ be interfering paths for robots i and j with critical sections $\mathcal{C}_{ij}$. The trajectories $\boldsymbol{p}_i(\sigma_i)$ and $\boldsymbol{p}_j(\sigma_j)$ will not collide if σ_i and σ_j adhere to the constraint

$$\langle \boldsymbol{p}_i, \boldsymbol{p}_j, \inf_{\boldsymbol{p}_i} C_{ij}, \sup_{\boldsymbol{p}_j} C_{ij} \rangle, \tag{2}$$

for each critical section $C_{ij} \in \mathcal{C}_{ij}$.

Each of the above constraints imposes the complete sequencing of robots through a critical section C_{ij}: while robot j has not exited the critical section, robot i is not allowed to enter it. Note that these constraints are very conservative: if the paths of the two robots through a critical section are not in opposing directions, then it may be possible for two robots to be in the critical section at the same time without colliding. For this, we require a more granular constraint, where the configuration q_i depends on the position of robot j in the critical section at time t. Let $\mathrm{reach}(\boldsymbol{p}_i, \boldsymbol{p}_j, t)$ indicate the latest configuration that is reachable by robot i along path $\boldsymbol{p}_i$ before its transformation overlaps with that of robot j at time t on path $\boldsymbol{p}_j$, that is:

$$\mathrm{reach}(\boldsymbol{p}_i, \boldsymbol{p}_j, t) = \sup_{\boldsymbol{p}_i} \{q \in \boldsymbol{p}_i^{[0,t]} \mid R_i(q) \cap R_j(\boldsymbol{p}_j(\sigma_j(t))) = \emptyset\}. \tag{3}$$

The condition expressed in Remark 1 can be relaxed as follows:

Remark 2. Let $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ be interfering paths for robots i and j with critical sections $\mathcal{C}_{ij}$. The trajectories $\boldsymbol{p}_i(\sigma_i)$ and $\boldsymbol{p}_j(\sigma_j)$ will not collide if σ_i and σ_j adhere to the constraint

$$\langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i(t), \sup_{\boldsymbol{p}_j} C_{ij} \rangle, \tag{4}$$

for each critical section $C_{ij} \in \mathcal{C}_{ij}$, where $q_i(t) =$

$$\begin{cases} \sup_{\boldsymbol{p}_i}\{\inf_{\boldsymbol{p}_i} C_{ij}, \mathrm{reach}(\boldsymbol{p}_i, \boldsymbol{p}_j, t)\}, & \text{if } \sup_{\boldsymbol{p}_j} C_{ij} \notin \boldsymbol{p}_j^{[0,t]} \\ \boldsymbol{p}_i(1), & \text{otherwise} \end{cases} \tag{5}$$

The constraints above impose that no robot i may proceed beyond the current location of any robot j, while robot j occupies critical section C_{ij}. Unlike those in Remark 1, the constraints formulated in Remark 2 are time-dependent precedence constraints, where the limit to which robot i is allowed to navigate is a function of the current progress of robot j, as stated in eq. (5). As explained in the next Section, this results in robots "following" each other through critical sections whenever possible (see Figure 2).

A set of precedence constraints $\mathcal{T}$ such that $\langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i(t), \sup_{\boldsymbol{p}_j} C_{ij} \rangle \in \mathcal{T}$ thus defines an order of traversal through critical section C_{ij} by robots i and j. We indicate this fact with the notation $(i <_{C_{ij}} j) \in \mathcal{T}$, reflecting the fact that the constraints give priority to j over i through critical section C_{ij}.

Definition 8. Let $\mathcal{C}$ be the set of all pairwise critical sections among the paths of an arbitrary number of robots. The set of precedence constraints $\mathcal{T}$ is a *complete ordering for robots* through $\mathcal{C}$ iff $(i <_{C_{ij}} j) \in \mathcal{T}$ or $(j <_{C_{ij}} i) \in \mathcal{T}$ for all $C_{ij} \in \mathcal{C}$.

By construction, a complete ordering for all robots eliminates the possibility of collisions:

Lemma 1. *Given a set $\mathcal{P}$ of paths for an arbitrary number of robots, the resulting set of all pairwise critical sections $\mathcal{C}$, and a complete ordering $\mathcal{T}$ through $\mathcal{C}$ containing constraints defined as in eqs.* (3) *to* (5)*, if the temporal profile $\sigma_i(t)$ of each robot trajectory $\boldsymbol{p}_i$ adheres to the constraints $\mathcal{T}$, then the robots will not collide.*

Proof. Follows directly from the fact that constraints $\mathcal{T}$ constitute a complete ordering through $\mathcal{C}$ and that all temporal profiles adhere to the constraints. □

Coordination Algorithm

Algorithm 1 realizes a high-level control loop for regulating access to critical sections for a fleet of robots, running at a given frequency $1/T$. At every iteration, the state of robots is sampled (line 5), and a path is computed leading each idle robot from its current configuration to the requested goal configuration (line 10). Critical sections are computed by obtaining the pairwise intersections of the spatial envelopes of all paths (line 13). These, along with the current state of the robots, are used to revise the set of constraints $\mathcal{T}$ to which the temporal profiles should be subject to. It is assumed that temporal profiles are computed/updated by the individual robot controllers upon calls to the `updateTrajectory` function. For each robot i, the algorithm assesses whether the constraints require it to yield in some configuration q_i (lines 16–17). If so, the closest such configuration is found, and provided to the controller along with its path $\boldsymbol{p}_i$ (lines 18–19). If a robot's trajectory is not subject to constraints, its controller is notified that it can proceed until the end of the current path $\boldsymbol{p}_i(1)$ (lines 20–21).

Algorithm 1: The `coordination` algorithm.

Input: a set G containing goals posted for robots $\{1, \ldots, n\}$.

1 $\mathcal{P} \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset$
2 **while** `true` **do**
3 $\quad t \leftarrow$ `getCurrentTime()`
4 $\quad$ **for** $i \in [1\,..\,n]$ **do**
5 $\quad\quad s_i \leftarrow$ `sampleState(`i`)`
6 $\quad$ **for** $i : g_i \in G \wedge$ `isIdle(`s_i`)` **do**
7 $\quad\quad G \leftarrow G \setminus \{g_i\}$
8 $\quad\quad$ remove all elements relative to robot i from $\mathcal{P}$ and $\mathcal{C}$
9 $\quad\quad q_i \leftarrow$ `getConfiguration(`s_i`)`
10 $\quad\quad \boldsymbol{p}_i \leftarrow$ `computePath(`q_i`,`g_i`)`
11 $\quad\quad \mathcal{P} \leftarrow \mathcal{P} \cup \{\boldsymbol{p}_i\}$
12 $\quad$ **for** $(\boldsymbol{p}_i, \boldsymbol{p}_{j \neq i}) \in \mathcal{P}^2$ **do**
13 $\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup$ `getIntersections(`$\mathcal{E}(\boldsymbol{p}_i), \mathcal{E}(\boldsymbol{p}_j)$`)`
14 $\quad \mathcal{T} \leftarrow$ `reviseConstraints(`$\mathcal{P}, \mathcal{C}, \mathcal{T}, t, \{s_1, \ldots, s_n\}$`)`
15 $\quad$ **for** $\boldsymbol{p}_i \in \mathcal{P}$ **do**
16 $\quad\quad T_i = \{q_i \mid \exists j : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i, q_j \rangle \in \mathcal{T}\}$
17 $\quad\quad$ **if** $T_i \neq \emptyset$ **then**
18 $\quad\quad\quad q_i^{\text{closest}} \leftarrow \arg\min_{q_i \in T_i} \boldsymbol{p}_i^{-1}(q_i)$
19 $\quad\quad\quad$ `updateTrajectory(`$\boldsymbol{p}_i$`,`q_i^{closest}`)`
20 $\quad\quad$ **else**
21 $\quad\quad\quad$ `updateTrajectory(`$\boldsymbol{p}_i$`,`$\boldsymbol{p}_i(1)$`)`
22 $\quad$ **while** `getCurrentTime()`$-t < T$ **do**
23 $\quad\quad$ `sleep(`Δt`)`

The core of the `coordination` Algorithm is procedure `reviseConstraints`, which decides if, when and where robots should yield. This is shown in Algorithm 2, which takes as input the current robot paths, the current time, and the current state of all robots. For each critical section, it assesses the state of the involved robots (lines 3, 4, 6, 8). Depending on the situation, it computes an ordering between the two robots involved in the critical section. If neither of the robots involved in a critical section have entered the critical section, then the choice of which robot should have access to the critical section first is decided by a function `computeOrdering` (line 5). As we show below, this function can be designed with more or less (or even no) knowledge of robot dynamics. Note that if either robot has navigated beyond a critical section, this will not lead to a constraint. The configuration beyond which the yielding robot should not navigate is computed by procedure `computeCriticalPoint` (line 10), which implements eqs. (3) to (5). The resulting revised set of constraints is returned to the coordination loop

and used to update robot trajectories as described above.

Algorithm 2: The `reviseConstraints` algorithm.

Input : a set $\mathcal{P}$ of paths for an arbitrary number of robots; a (possibly empty) set $\mathcal{C}$ of pairwise critical sections for $\mathcal{P}$; a (possibly empty) set $\mathcal{T}$ of precedence constraints; the current state s_i for each robot i; the current time t.

Output: a set of revised precedence constraints $\mathcal{T}_{\text{rev}}$.

1 $\mathcal{T}_{\text{rev}} \leftarrow \emptyset$
2 **for** $C_{ij} \in \mathcal{C}$ **do**
3 **if** $\sup_{\boldsymbol{p}_i} C_{ij} \notin \boldsymbol{p}_i^{[0,t]} \ \wedge \ \sup_{\boldsymbol{p}_j} C_{ij} \notin \boldsymbol{p}_j^{[0,t]}$ **then**
4 **if** $\inf_{\boldsymbol{p}_i} C_{ij} \notin \boldsymbol{p}_i^{[0,t]} \ \wedge \ \inf_{\boldsymbol{p}_j} C_{ij} \notin \boldsymbol{p}_j^{[0,t]}$ **then**
5 $(k, m) \leftarrow$ `computeOrdering`$(C_{ij}, \mathcal{T}, s_i, s_j)$
6 **else if** $\inf_{\boldsymbol{p}_i} C_{ij} \in \boldsymbol{p}_i^{[0,t]} \ \wedge \ \inf_{\boldsymbol{p}_j} C_{ij} \notin \boldsymbol{p}_j^{[0,t]}$ **then**
7 $(k, m) \leftarrow (i, j)$
8 **else if** $\inf_{\boldsymbol{p}_i} C_{ij} \notin \boldsymbol{p}_i^{[0,t]} \ \wedge \ \inf_{\boldsymbol{p}_j} C_{ij} \in \boldsymbol{p}_j^{[0,t]}$ **then**
9 $(k, m) \leftarrow (j, i)$
10 $q_m \leftarrow$ `computeCriticalPoint`$(\boldsymbol{p}_m, \boldsymbol{p}_k, t)$
11 $\mathcal{T}_{\text{rev}} \leftarrow \mathcal{T}_{\text{rev}} \cup \{\langle \boldsymbol{p}_m, \boldsymbol{p}_k, q_m, \sup_{\boldsymbol{p}_k} C_{ij} \rangle\}$
12 **return** $\mathcal{T}_{\text{rev}}$

The examples in Figure 2 show four moments during the execution of trajectories for two robots navigating through a long critical section, coordinated by Algorithm 1. An arrow from i to j indicates that $\mathcal{T}$ contains a precedence constraint $\langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i(t), \inf_{\boldsymbol{p}_i} C_{ij} \rangle$. The top row shows a first example in which the robots are navigating in opposing directions, which leads to one robot waiting until the other has completely cleared the critical section. In the bottom row, the robots are tasked to navigate from right to left along the shown paths. The constraint $\langle \boldsymbol{p}_1, \boldsymbol{p}_2, q_1(t), \inf_{\boldsymbol{p}_1} C_{12} \rangle$ is revised every T s, that is, $q_1(t)$ is updated to reflect the current state of robot 2 according to eq. (5). This brings about a "following" behavior, by which the critical point of robot 1 is continuously advanced while robot 2 progresses along $\boldsymbol{p}_2$.

Overall, the assumptions made in our approach are evident from the algorithm listings above: we assume that (1) kinematically-feasible reference paths can be computed via some motion planning method (line 10, Algorithm 1); (2) robot controllers are capable of reporting their current pose and whether or not they are idle (lines 5, 6, 9, Algorithm 1, line 5, Algorithm 2); and (3) robot controllers are capable of updating their reference trajectory with a new setpoint, namely, the critical point beyond which navigation is forbidden (lines 19, 21, Algorithm 1). We also assume that robot controllers do not lead robots to poses that lie outside spatial envelopes (which would invalidate Lemma 1).

A Simple Robot Ordering Policy

Let us assume that we have no knowledge of the dynamics of the robots in the fleet. We can formulate a simple robot ordering policy that decides a complete ordering whenever a new goal is posted, and never changes that ordering in subsequent iterations of the coordination loop. Given a critical section $C_{ij} \in \mathcal{C}$, the current states s_i and s_j of the involved robots, and the set of precedence constraints $\mathcal{T}$, the robot ordering (k, m) returned by `computeOrdering` is one of the two permutations $\{(i, j), (j, i)\}$, determined as follows:

$$(k, m) = \begin{cases} (i, j), & \text{if } (j <_{C_{ij}} i) \in \mathcal{T} \\ (j, i), & \text{if } (i <_{C_{ij}} j) \in \mathcal{T} \\ (k, m) : \dot{q}(s_m) = 0, & \text{otherwise} \end{cases} \quad (6)$$

where $\dot{q}(s_m)$ is the current speed of robot m. The policy above ensures that an ordering is decided only once, and never changed thereafter. This ordering is the permutation (k, m) such that an idle robot never has priority over a non-idle robot.

Theorem 1. *Algorithm 1 guarantees the absence of collisions if* `computeOrdering` *(Algorithm 2, line 5) decides the ordering of robots at each critical section as specified in eq.* (6)*.*

Proof. The ordering through every critical section will be imposed for the first time when at least one of the involved robots is idle (Algorithm 1, lines 6, 19, 21). It is guaranteed that robot m will be able to respect the resulting precedence constraint $\langle \boldsymbol{p}_m, \boldsymbol{p}_k, q_m, \sup_{\boldsymbol{p}_k} C_{ij} \rangle$ because m is idle (hence does not have to slow down). Algorithms 1 and 2 ensure that $\mathcal{T}$ is at all times a complete ordering of robots for critical sections $\mathcal{C}$, and that the closest yielding configuration along a robot's path is communicated to the controller (Algorithm 1, lines 18–19). Hence robots are guaranteed not to collide if the temporal profiles they compute when receiving a trajectory update ensure that the critical point is reached with zero velocity and not passed (Lemma 1). □

Heuristic Robot Ordering Policies

It may be beneficial to base ordering decisions on other factors (e.g., heuristics that minimize an objective function). Also, it may be useful to change robot orderings depending on the observed performance of the fleet, potentially at every period T. In order to do this, it is required to assess the physical realizability of ordering decisions for robots in motion considering their dynamics. Let

$$\ddot{q}_i^{t+\Delta t} \approx g_i(q_i^t, \dot{q}_i^t, u_i^t), \quad (7)$$

$$\dot{q}_i^{t+\Delta t} \approx \dot{q}_i^t + \ddot{q}_i^{t+\Delta t} \Delta t, \quad (8)$$

$$q_i^{t+\Delta t} \approx q_i^t + \dot{q}_i^{t+\Delta t} \Delta t, \quad (9)$$

be a forward model of the dynamics of robot i, where u_i^t is an appropriately formed (set of) control(s) at time t. Let $(i <_{C_{ij}} j) \in \mathcal{T}$, and let u_i^{dec} be the maximum deceleration control that can be given to robot i. Assuming the forward model is conservative, given the state s_i of robot i, we can compute $\hat{t} : \dot{q}_i^{\hat{t}} = 0$ by extrapolation from eqs. (7) to (9) with $q_i^0 = q(s_i)$, $\dot{q}_i^0 = \dot{q}(s_i)$, and $u_i^t = u_i^{\text{dec}}$. If the resulting position of robot i at time $\hat{t}$ is not beyond the beginning of critical section C_{ij},

$$\boldsymbol{p}_i^{-1}(q_i^{\hat{t}}) < \boldsymbol{p}_i^{-1}(\inf_{\boldsymbol{p}_i} C_{ij}), \quad (10)$$

then it is possible for robot i to yield for robot j at critical section C_{ij}.

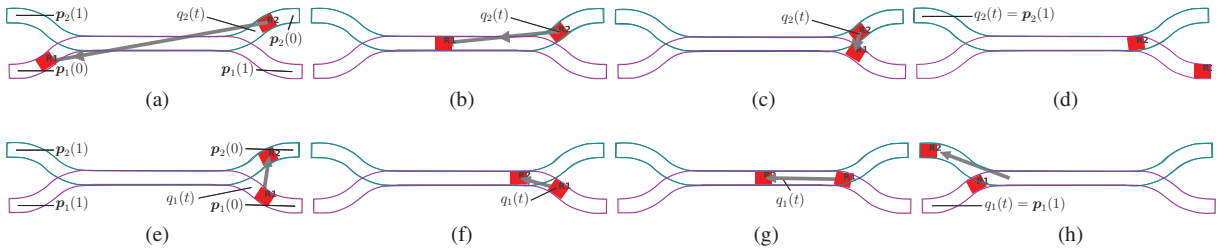

Figure 2: Two examples of robots yielding to each other: (a–d) four moments during navigation in opposing directions; (e–h) four moments during navigation in the same direction. An arrow from i to j indicates that robot i's critical point depends on the progress of robot j, that is, $\mathcal{T}$ contains constraint $\langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i(t), \sup_{\boldsymbol{p}_j} C_{ij} \rangle$ at time t.

We can exploit conservative forward models to obtain a dynamic ordering policy that minimizes a heuristic function h. Given a critical section C_{ij}, the current states s_i and s_j of the robots, and the set of precedence constraints $\mathcal{T}$, let F_{ij} be a set of pairs such that $(j, i) \in F_{ij}$ iff eq. (10) holds, that is, robot i can come to a stop before entering critical section C_{ij}. The robot ordering (k, m) returned by `computeOrdering` for a critical section C_{ij} is

$$(k, m) = \underset{(k,m) \in F_{ij}}{\arg\min}\ h(s_k, s_m, k, m). \tag{11}$$

It is easy to see that any such heuristic ordering policy guarantees the absence of collisions:

Theorem 2. *Algorithm 1 guarantees the absence of collisions if* `computeOrdering` *(Algorithm 2, line 5) decides the ordering of robots at each critical section as specified in eqs.* (10) *and* (11) *with conservative forward models of robot dynamics.*

Proof. For each critical section, `computeOrdering` is invoked for the first time when at least one of the two involved robots is idle. Hence, it is guaranteed that a physically realizable ordering is computed at least once. The trajectory of the yielding robot is updated for the first time while the robot is idle (Algorithm 1, line 19). Therefore, this ordering will remain feasible as time goes by, and will be considered as a possible ordering in future invocations of `computeOrdering`. A future invocation of `computeOrdering` may change the first ordering, but this will occur only if neither robot has entered the critical section (Algorithm 2, line 4), and the new ordering is physically realizable according to the forward model. Algorithms 1 and 2 ensure that $\mathcal{T}$ is at all times a complete ordering of robots for critical sections $\mathcal{C}$, and that the closest yielding configuration along a robot's path is communicated to the controller (Algorithm 1, lines 18–19). Therefore, robots will not collide in virtue of Lemma 1. □

A plausible heuristic h is one that estimates the effect of the proposed ordering (k, m) on the total time to completion of all paths. Time to completion of a robot is proportional to the amount of time the robot spends yielding to other robots — the less yielding, the smoother the performance of the fleet and consequently the lower the overall time to completion. This suggests that a good heuristic for minimizing overall time to completion is the distance heuristic

$$h_{\text{dist}}(s_k, s_m, k, m) \equiv \boldsymbol{p}_k^{-1}(\inf_{\boldsymbol{p}_k} C_{km}) - \boldsymbol{p}_k^{-1}(q(s_k)),$$

which gives precedence to the robot that is closest to the beginning of a critical section. This effectively minimizes waiting time, because it avoids that robots traveling in opposing directions yield to each other unnecessarily. Other heuristics can be envisaged, e.g., functions that make use of the precedence constraints in $\mathcal{T}$, or that consider forward models to account for best-, average- and worst-case performance.

Deadlocks

We consider the issue of liveness, i.e., whether it is guaranteed that robots will always reach their goals.

Definition 9. Let $\mathcal{T}$ be a set of precedence constraints over robot paths $\mathcal{P}$ with critical sections $\mathcal{C}$, and let $D_{\mathcal{T}} = (V, E)$ be the dependency graph of the constraints, namely:

$$V = \{i \mid \boldsymbol{p}_i \in \mathcal{P}\},$$
$$E = \{(i, j) \in V^2 \mid \exists C_{ij} \in \mathcal{C} : (j <_{C_{ij}} i) \in \mathcal{T}\}.$$

If $D_{\mathcal{T}}$ contains a cycle $\langle i_1, \ldots, i_m = i_1 \rangle$, then $\mathcal{T}$ contains precedence constraints

$$\begin{gathered} \langle \boldsymbol{p}_{i_1}, \boldsymbol{p}_{i_2}, q'_{i_1}, q_{i_2} \rangle, \\ \langle \boldsymbol{p}_{i_2}, \boldsymbol{p}_{i_3}, q'_{i_2}, q_{i_3} \rangle, \\ \ldots \\ \langle \boldsymbol{p}_{i_{m-1}}, \boldsymbol{p}_{i_m}, q'_{i_{m-1}}, q_{i_m} \rangle. \end{gathered}$$

The cycle is *unsafe* iff $\boldsymbol{p}_{i_j}^{-1}(q_{i_j}) > \boldsymbol{p}_{i_j}^{-1}(q'_{i_j})$ for all $j \in [2\,..\,m]$.

If there are no unsafe cycles, there is at least one robot that does not have to wait for another robot to enter a critical section. Once a robot has entered a critical section, Algorithm 2 never changes the priority of that robot through the critical section (lines 6–9). As a consequence, if no robot path terminates in a critical section, then any robot in a critical section will eventually exit that critical section. That is,

Remark 3. If $\mathcal{T}$ contains no unsafe cycles and robots are not in critical sections at the beginning and end of their paths, then any set of temporal profiles that satisfies $\mathcal{T}$ is deadlock-free.

Assuming that $|G| \leq 1$ at each iteration (at most one goal is posted per period in Algorithm 1), the simple ordering policy described in eq. (6) ensures the absence of cycles, as this entails that any other robot will have priority over the robot with a new goal, and this ordering will not be changed

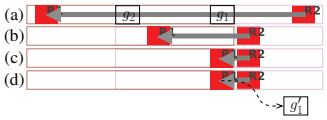

Figure 3: The paths leading R1 to reach g_1 and R2 to reach g_2 cause a deadlock (a–c); when a new goal g_1' is posted for R1, the path computed to reach it may again lead to a deadlock, unless motion planning is performed considering the two robots jointly.

in subsequent iterations. This effectively limits the scheduling of robots through critical sections to a fixed ordering, determined entirely by the order in which goals are posted.

The robot ordering policy in eqs. (10) and (11) does not guarantee the absence of cycles, even assuming one goal per period, as orderings can be changed while robots navigate, based on the heuristic function h. In order to guarantee the absence of deadlocks, we therefore need to ensure that any cycle that may appear in $\mathcal{T}$ is safe. This is problematic if we relax the assumption that robots are not in critical sections at the beginning and end of their paths. For instance, given a critical section C_{ij} such that $\boldsymbol{p}_i^{-1}(\sup_{\boldsymbol{p}_i} C_{ij}) = 1$ (robot i "parks" inside critical section C_{ij}), allowing robot j to precede robot i may avert the deadlock.[1] This will not resolve the deadlock if robot j's path also terminates within the critical section. We may rely on the fact that robot j will eventually receive a new goal — however, the newly computed path may also lead to a deadlock (e.g., see Figure 3).

It is worth noting that there can be several strategies for attempting deadlock resolution. One is to allow robots to backtrack along their current trajectory to the first pose that is not in a critical section, assuming one exists. An alternative strategy is to consider possible placements of other robots during path computation: when computing the path for robot i, consider an obstacle $R_j(q_j)$ for each robot $j \neq i$ in configuration q_j such that $\langle \boldsymbol{p}_j, \boldsymbol{p}_i, q_j, q_i \rangle \in \mathcal{T}$. A similar form of prioritized planning is described by Čáp et al. (2015), which averts deadlocks if the environment is "well-formed", a notion that corresponds to a specific case of our earlier assumption that $\boldsymbol{p}_i^{-1}(\inf_{\boldsymbol{p}_i} C_{ij}) > 0$ and $\boldsymbol{p}_i^{-1}(\sup_{\boldsymbol{p}_i} C_{ij}) < 1, \forall i, j$.

Evaluation

We evaluate our coordination algorithm from three points of view. Experiment one seeks to assess the scalability of the system. Experiment two analyzes the impact of the coordination period T on fleet performance. Experiment three shows how coordination copes with the uncertainties of real robots moving in a physical environment, and how coordination copes with unforeseen contingencies.

The experiments also demonstrate the use of the `coordination` algorithm with different motion planners and robot controllers. In the first experiment, a low-fidelity simulation back-end is used, and paths are pre-planned and

[1]This strategy is similar to that of adding null-segments in the collision regions of task completion diagrams described by O'Donnell and Lozano-Perez (1989).

Figure 4: A moment during the execution of the first experiment (only 5 robots shown for clarity).

loaded from a library; a slightly higher-fidelity simulator is used in the second experiment, which accounts for simple robot dynamics, and paths are computed on the fly with an off-the-shelf RRT-based motion planner. In the third experiment, we use a lattice-based motion planner and real robots driven by an MPC-based control scheme. Simulated robot controllers (experiments 1 and 2) employed a simple trapezoidal velocity profile, with constant acceleration/deceleration of $\pm 3\,\mathrm{m\,s^{-2}}$ and maximum velocity of $14\,\mathrm{m\,s^{-1}}$ (50.4 kph). In all experiments, the h_{dist} heuristic was used, and a linear acceleration model was used as forward model for coordination. The `coordination` Algorithm, which ran on a 4-core, 8-thread Intel Core i7-6700HQ CPU, is implemented in Java and available as open source (Pecora, 2017a). The `getIntersections` procedure is realized with the JTS computational geometry library (LocationTech, 2012). A summary video of experiments 2 and 3 is available online (https://youtu.be/jCgrCVWf8sE).

Scalability

Two tests were performed to evaluate scalability. The first was designed to provoke frequent critical section computations (lines 12–13, Algorithm 1). We simulated a fleet of 50 robots, with each robot i initially parked in position A_i as shown in Figure 4. Every 20 seconds, one additional robot was posted the goal to reach the corresponding position B_i along a precomupted path (spatial envelopes shown in the figure). Whenever a robot reached position B_i, a new goal to reach position A_i was immediately posted (and vice-versa). Thus, after $n \times 20$ seconds, at most n robots were in motion. As all paths overlap in a choke point, the addition of the n-th robot required computing a further $(n-1)$ critical sections.

The `coordination` algorithm was run with $T = 0\,\mathrm{s}$, i.e., the period length was the time needed to perform all operations in one iteration of the outer loop. Figure 5 plots the average period length over all iterations against number of robots in motion. At most 23 robots were in motion at a time (idle robots were in the process of receiving a new trajectory while in position A_i or B_i). The results show that even with significant critical section computation overhead, coordination never requires more than $0.5\,\mathrm{s}$ per iteration.

The second test was designed to measure the computational overhead of constraint revision and trajectory updating (lines 14–21, Algorithm 1). For this purpose, we placed 30 robots in poses A_i as shown in Figure 6, and posted 30 goals simultaneously, one for each robot i, to reach pose B_i along the shown sinusoidal path. Whenever a robot reached B_i, a new goal to reach A_i along the inverse path was immediately posted (and vice-versa).

Figure 7 plots period length and number of driving robots at each iteration. When many robots are in motion, frequent yielding behavior is observed, as critical sections are spread

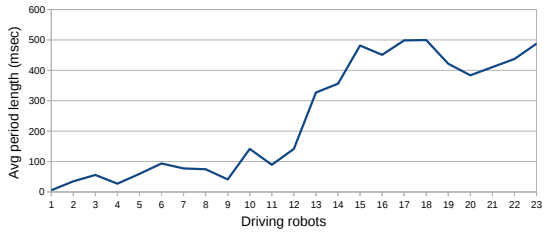

Figure 5: Average period length (in milliseconds) against number of robots in motion with a fleet of 50 robots.

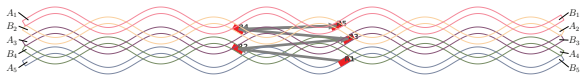

Figure 6: A moment during the execution of the second experiment (only 5 robots shown for clarity).

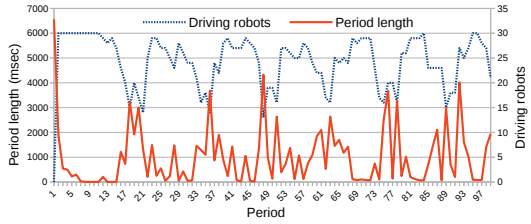

Figure 7: Period length (in milliseconds) against number of robots in motion with a fleet of 30 robots.

all along the robot paths. However, the computational overhead is low, as no or few new critical sections are computed, and revising constraints and updating trajectories require linear time in the number of robots in motion.

Effect of Coordination Period

The lower the period T can be set, the more often constraints can be revised and critical points communicated to the robots, leading to smoother coordinated movements and better time to completion of trajectories. However, communicating often may not be possible in real situations. This impacts time to completion: robot controllers will adjust navigation speed to account for dynamics, and revising constraints less frequently may lead to missed opportunities to let faster robots gain access to critical sections. (Recall that coordination does not use the forward model to predict *when* a robot will reach a critical section, but only to assess *whether it is possible* for a robot to yield at a critical section.)

We performed an experiment (of which we omit the details for brevity) to assess the effect of T on time to completion. The test involved five simulated robots whose controllers slow the robots down along path segments with high curvature. The robots were continuously posted goals in an environment with obstacles, and paths were computed on the fly with an off-the-shelf RRT-Connect single-query motion planner (Kuffner and LaValle, 2000). The fact that some robots needed to slow down along their path (for reasons other than coordination) led to frequent occasions in which h_{dist} could "change its mind" on which robot was given priority through a critical section. The coordination period was varied between $T = 0.2\,\text{s}$ and $T = 3\,\text{s}$, and average time to completion was computed over 20 completed paths, for each value of T. As expected, we observed that updating critical points less frequently leads to worse time to completion of trajectories. However, the increase in time to completion is linear, indicating that the performance of the fleet does not decrease drastically when the frequency of updates is reduced. A video of the test with $T = 0.2\,\text{s}$ is available online (https://youtu.be/45rulULlFPs).

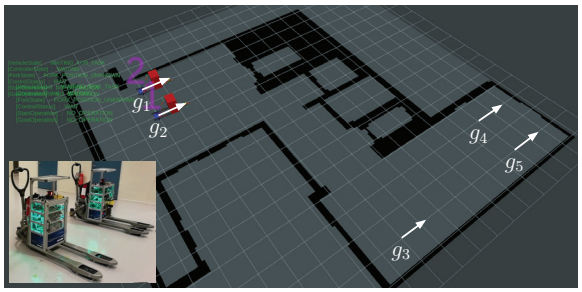

Figure 8: RViz view of the two robots in the basement. Robot 1 is continuously posted goals $\{g_3, g_4, g_2\}$ in sequence, while robot 2 cycles through goals $\{g_3, g_5, g_1\}$. The two robots used in the experiment are shown in the inset.

Dealing with Real Robots and Contingencies

We now describe a realization of the approach on a real multi-robot system. Two autonomous forklifts (Linde Cititrucks) are placed in a $100\,\text{m}^2$ basement with a corridor connecting two areas, see Figure 8. Each robot is controlled by an on-board MPC controller. MPC-based control is well suited for a wide variety of industrial applications (Qin and Badgwell, 2003), and in particular for non-holonomic robot systems (Zhu and Özgüner, 2008) like the forklifts used in this experiment. The robots are localized with an industrial-grade laser-based system that uses static reflective beacons as landmarks. Each time a goal was posted, a path to reach it from the robot's current pose was computed via a lattice-based motion planner (Andreasson et al., 2015). All motion planning and control algorithms were run on the robots, while the `coordination` algorithm ran on a laptop connected to the fleet via a ROS interface (Pecora, 2017b).

In the experiment, the robots start off placed in poses g_1 and g_2, as shown in Figure 8. We performed a first test to assess the general performance of the method in this physical robot setup. The robots successfully performed ten cycles, visiting all goals and yielding at critical sections. Frequent coordination was required in the corridor connecting the two larger spaces (navigating from $\{g_1, g_2\}$ to g_3 and from g_5 back to $\{g_1, g_2\}$), as well as when both robots maneuvered around goals $\{g_3, g_4, g_5\}$. A video of the entire experiment is available online (https://youtu.be/OFin8SYAsvM).

A second test was performed to verify the behavior of the system in the presence of contingencies, and to un-

derscore the benefit of continuous constraint revision (using h_{dist}). While both robots were navigating around goals $\{g_3, g_4, g_5\}$, a low-level brake command was sent to the robot controllers, causing the robots to stop moving and to enter an error state. Nothing was communicated to the coordinator, which continued to receive feedback only in the form of the current pose of robots via invocations of `sampleState` (line 5, Algorithm 1). Robot 2 was then allowed to recover from the error state. The heuristic had initially determined that robot 1 should have precedence in accessing the corridor. However, since robot 1 failed to advance further (as it remained stuck in an error state), robot 2 was given priority upon a subsequent constraint revision, leading it to overtake robot 1. A video of the test is available online (https://youtu.be/dmPUoLuRUn8).

Related Work

The Operations Research and Multi-Agent Systems communities have focused on the Multi-Agent Path Finding (MAPF) problem, that is, to find the paths for multiple agents in a given graph from their current vertices to goal vertices without colliding with other agents, while optimizing a cost function. Solutions to MAPF scale to hundreds of agents with reasonable boundaries on optimality. However, unrealistic assumptions are often made on the agents and their motions, e.g., point-shaped robots, with little or no kinodynamic considerations, moving in a simplified grid representation of the environment (Ma et al., 2017). The use of these solutions for automation is therefore limited to environments that can be engineered to suit these assumptions, e.g., warehouses (Wurman, D'Andrea, and Mountz, 2007).

Graph-based search methods have been employed with some success for coordination. Here, the environment is represented as a graph, whose connectivity represents the possible motions of robots. Conceived for maze-like, congested environments, these methods are not designed for online readjustment (Wagner and Choset, 2013; Sharon et al., 2015). It is also assumed that all robots start their missions at the same time, and trajectories are equated to paths through the graph, hence are piece-wise continuous. The latter assumption requires robots to stop at each traversed node to ensure dynamic feasibility. Preiss et al. (2017) and Hönig et al. (2016) post-process the resulting collision-free paths to transfer them to kinodynamically feasible trajectories.

The Robotics literature reports a variety of approaches for multi-robot motion planning. Most account for robot-robot collisions via a joint configuration space derived from the Cartesian product of the configuration spaces of all robots (LaValle, 2006). This is computationally expensive for large fleets, and not adequate for online use. Some address the coordination problem in coordination space (O'Donnell and Lozano-Perez, 1989), whose points represent the progress of all robots along their trajectories. Using this concept, a provably collision- and deadlock-free control law for multi-robot systems was proposed by Čáp, Gregoire, and Frazzoli (2016). The approach, however, assumes holonomic, disc-shaped robots, and requires all paths to be known in order to compute the coordination space (whereas in our approach, goals can be posted dynamically).

The multi-robot motion planning problem can also be formulated in a continuous domain, and continuous-variable optimization can be used to find feasible trajectories for multiple robots. These approaches produce smooth and feasible trajectories for vehicles with non-trivial kinematics (Augugliaro, Schoellig, and D'Andrea, 2012; Deits and Tedrake, 2015). However, solutions scale badly to a large fleets, due to the complexity of the optimization problem. Kamel et al. (2017) formulate an online coordination problem for Multiple Micro Aerial Vehicles (MAVs) as an optimal control problem with receding horizon. The approach is decentralized, and accurately accounts for kinodynamic constraints and uncertainty on MAV position estimates. However, scalability remains under-addressed.

Our approach to multi-robot coordination relies on posting critical points along trajectories to avoid collisions. In a similar vein, Peng and Akella (2005) adjust the speed profiles of robots along specified paths to avoid robot-robot collisions. Unlike our approach, an optimal control strategy is employed for computing kinodynamically-feasible trajectories, and collision avoidance constraints for pairs of robots are upheld by solving a mixed-integer nonlinear programming problem. Bareiss and Van den Berg (2013) propose a similar approach for coordinating the motions of multiple robots with predefined paths. In both works, the collision avoidance problem is formulated online, and it is not assumed that robots start moving concurrently. However, these approaches require to formulate collision-avoidance constraints in the robot controllers, so that kinodynamically-feasible and collision-free target velocities can be computed online without explicit coordination. Also, neither approach includes global path planning, i.e., robots are assumed to move in an obstacle-free environment.

Conclusions and Future Work

We have presented a method for coordinating fleets of autonomous robots that can be used with off-the-shelf motion planning and control modules. We minimize the assumptions made on these modules, requiring only that robot controllers can commit to dynamically feasible set-point updates. We have shown through formal analysis that the method is sound, discussed how existing deadlock-avoidance strategies can be included, and evaluated the approach with simulated and real robot systems.

It is worth noting that spatial envelopes can be defined as the set of robot transformations that *can be* reached along the path, as opposed to those that are reached under the assumption that the robot controller will follow the reference path perfectly. For this purpose, spatial envelopes can be generalized to flow tubes (Li and Williams, 2008) or trajectory envelopes (Pecora, Cirillo, and Dimitrov, 2012). The notion of interference would then take on the meaning of "possible interference", capturing the fact that robot controllers can be given bounds on how much they are allowed to stray from the reference path. Adapting the `coordination` Algorithm to alternative definitions of envelopes reduces to redefining the `computeIntersections` procedure and generalizing the notion of critical point (to, e.g., spatial constraints beyond

which a robot is not allowed to navigate). We will address this issue in future work.

Acknowledgments. This work is supported by the Semantic Robots Research Profile, funded by the Swedish Knowledge Foundation (KKS), EU Project ILIAD (GA no. 732737), and Vinnova project iQMobility.

References

Andreasson, H.; Saarinen, J.; Cirillo, M.; Stoyanov, T.; and Lilienthal, A. J. 2015. Fast, continuous state path smoothing to improve navigation accuracy. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*.

Augugliaro, F.; Schoellig, A. P.; and D'Andrea, R. 2012. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Bareiss, D., and Van den Berg, J. 2013. Reciprocal collision avoidance for robots with linear dynamics using lqr-obstacles. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*.

Chung, W. K.; Fu, L.-C.; and Kröger, T. 2016. Motion control. In Siciliano, B., and Khatib, O., eds., *Springer Handbook of Robotics*. Cham: Springer International Publishing. 163–194.

Deits, R., and Tedrake, R. 2015. Efficient mixed-integer planning for uavs in cluttered environments. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*.

Elbanhawi, M., and Simic, M. 2014. Sampling-based robot motion planning: A review. *IEEE Access* 2:56–77.

Hönig, W.; Kumar, T. K. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-agent path finding with kinematic constraints. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Kamel, M.; Alonso-Mora, J.; Siegwart, R.; and Nieto, J. 2017. Robust collision avoidance for multiple micro aerial vehicles using nonlinear model predictive control. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Kuffner, J., and LaValle, S. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*.

LaValle, S. M. 2006. *Planning Algorithms*. New York, NY, USA: Cambridge University Press.

Li, H. X., and Williams, B. C. 2008. Generative planning for hybrid systems based on flow tubes. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*.

LocationTech. 2012. JTS topology suite (version 1.13). https://github.com/locationtech/jts.

Ma, H.; Li, J.; Kumar, T. S.; and Koenig, S. 2017. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proc. of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 837–845.

O'Donnell, P., and Lozano-Perez, T. 1989. Deadlock-free and collision-free coordination of two robot manipulators. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*.

Pecora, F.; Cirillo, M.; and Dimitrov, D. 2012. On mission-dependent coordination of multiple vehicles under spatial and temporal constraints. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Pecora, F. 2017a. An online multi-robot coordination algorithm based on trajectory envelopes (version 0.1.1). Source code available at http://github.com/FedericoPecora/coordination_oru.

Pecora, F. 2017b. The `coordination_oru_ros` multi-robot coordination package (version 0.1.0). Source code available at http://github.com/FedericoPecora/coordination_oru_ros.

Peng, J., and Akella, S. 2005. Coordinating multiple robots with kinodynamic constraints along specified paths. *International Journal of Robotics Research* 24(4):295–310.

Preiss, J. A.; Hönig, W.; Ayanian, N.; and Sukhatme, G. S. 2017. Downwash-aware trajectory planning for large quadrotor teams. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Qin, S., and Badgwell, T. 2003. A survey of industrial model predictive control technology. *Control Engineering Practice* 11:733–764.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.* 219(C):40–66.

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

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

Wagner, G., and Choset, H. 2013. M*: A complete multirobot path planning algorithm with optimality bounds. In Milutinović, D., and Rosen, J., eds., *Redundancy in Robot Manipulators and Multi-Robot Systems*. Berlin, Heidelberg: Springer. 167–181.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2007. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *Proc. of the 19th National Conference on Innovative Applications of Artificial Intelligence*, IAAI'07, 1752–1759.

Zhu, Y., and Özgüner, U. 2008. Constrained Model Predictive Control for Nonholonomic Vehicle Regulation Problem. In *17th World Congress IFAC*, 9552–9557.