Intra-Logistics with Integrated Automatic Deployment:
Safe and Scalable Fleets in Shared Spaces

H2020-ICT-2016-2017
Grant agreement no: 732737

DELIVERABLE 5.1

Report on optimization strategies for task allocation and coordination

Due date: month 12 (December 2017)
Deliverable type: R
Lead beneficiary: ORU

Dissemination Level: PUBLIC

Main author: Federico Pecora, João Salvado, Masoumeh Mansouri, Henrik Andreasson (ORU)

Contents

1 Introduction

Task allocation, motion planning, coordination and control are all essential for deploying fleets of autonomous robots. These four problems are intrinsically dependent: robot motions must be physically realizable by controls computed by robot controllers; they must also be coordinated in order to avoid collisions and deadlocks; and the motion feasibility, coordination and control problems are determined by the particular allocation of robots to goals decided in task allocation.

This document defines the overall fleet management problem, and relates it to several sub-problems that must be solved to determine which robots serve which tasks and how they jointly navigate in the environment. We assume that robots are placed in an environment with obstacles, which results in each robot r_i having a *configuration space* $\mathcal{Q}_i = \mathcal{Q}_i^{\text{free}} \cup \mathcal{Q}_i^{\text{obs}}$. We also assume that the path $\boldsymbol{p}_i$ of a robot r_i is a sequence of configurations in $\mathcal{Q}_i$. A robot can navigate over a path according to a temporal profile $\sigma_i(t)$, which determines the configuration $\boldsymbol{p}_i(\sigma(t))$ of robot r_i along $\boldsymbol{p}_i$ at time t. The overall fleet management problem can be defined as the following set of sub-problems:

Definition 1. Given a set $R = \{r_1, \ldots, r_n\}$ of robots, a set $G = \{g_1, \ldots, g_m\}$ of goal poses, and (joint) configuration space $\mathcal{Q} = \bigcup_i \left(\mathcal{Q}_i^{\text{obs}} \cup \mathcal{Q}_i^{\text{free}}\right)$, the *fleet management problem* is the problem of determining

1. a total function $a : G \to R$ associating goals to robots (goal allocation);
2. a path $\boldsymbol{p}_i$ for each robot r_i leading it from its current pose to the allocated goal pose $a^{-1}(r_i)$, and such that $\boldsymbol{p}_i \cap \mathcal{Q}_i^{\text{obs}} = \emptyset$ (motion planning);
3. a temporal profile $\sigma_i(t)$ for each robot path $\boldsymbol{p}_i$ such that $\boldsymbol{p}_i(\sigma(t))$ does not intersect $\boldsymbol{p}_{j\neq i}(\sigma(t))$ for all t (coordination);
4. a control policy $\pi_i(t) = \boldsymbol{u}_i(t)$ for each robot r_i, which determines control inputs that lead r_i to be in pose $\boldsymbol{p}_i(\sigma(t))$ at time t (control).

We outline in the sections that follow two solutions to the fleet management problem that have been studied during the first year of the ILIAD project: one that relies on a composition of loosely-coupled modules for solving the four sub-problems listed above, and one in which all sub-problems are formulated as one overall optimization problem.

2 A Loosely-Coupled Approach

In this section we present a lightweight coordination method that implements a high-level controller for a fleet of potentially heterogeneous robots. The rationale for developing this approach is grounded in the observation that the application-specific features of the environment and robots often narrow down the possible goal allocation, motion planning, and control methods that can be used. The approach is therefore *loosely-coupled*, in the sense that it relies on off-the-shelf approaches for solving the goal allocation, motion planning, and control problems. As we show below, the approach imposes very few assumptions 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.

2.1 Spatial Envelopes and Critical Points

Let $\mathcal{Q}$ be a configuration space, and let $\boldsymbol{p} : [0,1] \to \mathcal{Q}$ 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.

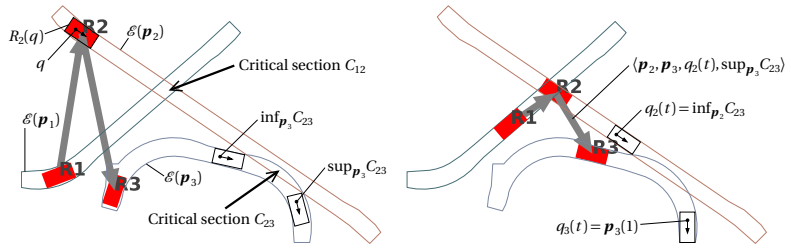


Figure 1: Three robots navigating along paths $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, and $\boldsymbol{p}_3$. Spatial envelopes and critical sections are shown on the left; gray arrows indicate precedence constraints; detail of the precedence constraint regulating robots 2 and 3 as they navigate through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

Let $R_j(q)$ be the transformation of robot j in a configuration $q \in \mathcal{Q}$ (see Figure 1, top). 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 2. Given a robot j with configuration space $\mathcal{Q}_j$, obstacles $\mathcal{O}$, 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:

Definition 3. 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 in Section 4), as they increase the computational effort required to solve the control problem. In the loosely-coupled approach, 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 that should be abided by in order to avoid collisions.

Definition 4. 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 5. 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 6. Given two paths $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, let $\mathscr{S} = \{q \in \mathscr{Q} \mid R_i(q) \cap \mathscr{E}(\boldsymbol{p}_j) \neq \emptyset \vee R_j(q) \cap \mathscr{E}(\boldsymbol{p}_i) \neq \emptyset\}$, and let $\mathscr{C}_{ij}$ be the decomposition of $\mathscr{S}$ into its largest contiguous subsets. Each set of configurations $C_{ij} \in \mathscr{C}_{ij}$ is called a *critical section*.

It follows from Definition 6 that $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ interfere if and only if $\mathscr{C}_{ij} \neq \emptyset$. Equivalently (see Definitions 4 and 5), $\boldsymbol{p}_i$ 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 $C_{ij} \in \mathscr{C}_{ij}$ 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 $\boldsymbol{p}_i(\inf_{\boldsymbol{p}_i} C_{ij}) > \boldsymbol{p}_i(0)$ and $\boldsymbol{p}_i(\sup_{\boldsymbol{p}_i} C_{ij}) < \boldsymbol{p}_i(1)$ for all i, that is, no robot begins or terminates its motion in a critical section. This assumption and the circumstances under which it can be relaxed are further discussed when we consider deadlocks.

Definition 7. 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 8. Given a set of paths $\mathscr{P}$ for an arbitrary number of robots, the *coordination problem* is the problem of synthesizing, for each pair of interfering paths $(\boldsymbol{p}_i, \boldsymbol{p}_{j \neq i}) \in \mathscr{P}^2$, 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 $\mathscr{C}_{ij}$. 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 $C_{ij} \in \mathscr{C}_{ij}$.

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 $\text{reach}(\boldsymbol{p}_i, \boldsymbol{p}_j, t)$ 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:

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

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 $\mathscr{C}_{ij}$. 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 $C_{ij} \in \mathscr{C}_{ij}$, where

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

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

Definition 9. Let $\mathscr{C}$ be the set of all pairwise critical sections among the paths of an arbitrary number of robots. The set of precedence constraints $\mathscr{T}$ is a *complete ordering for robots* through $\mathscr{C}$ iff $(i <_{C_{ij}} j) \in \mathscr{T}$ or $(j <_{C_{ij}} i) \in \mathscr{T}$ for all $C_{ij} \in \mathscr{C}$.

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

Lemma 1. *Given a set $\mathscr{P}$ of paths for an arbitrary number of robots, the resulting set of all pairwise critical sections $\mathscr{C}$, and a complete ordering $\mathscr{T}$ through $\mathscr{C}$ 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 $\mathscr{T}$, then the robots will not collide.*

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

2.2 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 $\mathscr{T}$ 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. If the precedence constraints call for a robot i to yield, then its path $\boldsymbol{p}_i$ is provided to the controller, along with the latest admissible configuration that can be reached by the robot (line 16). 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 $\boldsymbol{p}_i(1)$ (line 19).

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

Algorithm 1: The `coordination` algorithm.

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

$\mathscr{P} \leftarrow \emptyset, \mathscr{P}_{\text{constrained}} \leftarrow \emptyset, \mathscr{C} \leftarrow \emptyset, \mathscr{T} \leftarrow \emptyset$
while `true` **do**
  $t \leftarrow$ `getCurrentTime()`
  for $i \in [1..n]$ **do**
    $s_i \leftarrow$ `sampleState(`i`)`
  for $i : g_i \in G \wedge$ `isIdle(`s_i`)` **do**
    $G \leftarrow G \setminus \{g_i\}$
    remove all elements relative to robot i from $\mathscr{P}, \mathscr{P}_{\text{constrained}}, \mathscr{C}$
    $q_i \leftarrow$ `getConfiguration(`s_i`)`
    $\boldsymbol{p}_i \leftarrow$ `computePath(`q_i, g_i`)`
    $\mathscr{P} \leftarrow \mathscr{P} \cup \{\boldsymbol{p}_i\}$
  for $(\boldsymbol{p}_i, \boldsymbol{p}_{j\neq i}) \in \mathscr{P}^2$ **do**
    $\mathscr{C} \leftarrow \mathscr{C} \cup$ `getIntersections(`$\mathscr{E}(\boldsymbol{p}_i), \mathscr{E}(\boldsymbol{p}_j)$`)`
  $\mathscr{T} \leftarrow$ `reviseConstraints(`$\mathscr{P}, \mathscr{C}, \mathscr{T}, t, \{s_1,\dots,s_n\}$`)`
  for $i : \langle \boldsymbol{p}_i, \boldsymbol{p}_j, q_i, q_j \rangle \in \mathscr{T}$ **do**
    `updateTrajectory(`$\boldsymbol{p}_i, q_i$`)`
    $\mathscr{P}_{\text{constrained}} \leftarrow \mathscr{P}_{\text{constrained}} \cup \{\boldsymbol{p}_i\}$
  for $\boldsymbol{p}_i \in \mathscr{P} \setminus \mathscr{P}_{\text{constrained}}$ **do**
    `updateTrajectory(`$\boldsymbol{p}_i, \boldsymbol{p}_i(1)$`)`
  while `getCurrentTime()`$-t < T$ **do**
    `sleep(`Δt`)`

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

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

1 $\mathscr{T}_{\text{rev}} \leftarrow \emptyset$
2 **for** $C_{ij} \in \mathscr{C}$ **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`$(C_{ij}, \mathscr{T}, s_i, s_j)$
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 $\mathscr{T}_{\text{rev}} \leftarrow \mathscr{T}_{\text{rev}} \cup \{\langle \boldsymbol{p}_m, \boldsymbol{p}_k, q_m, \sup_{\boldsymbol{p}_k} C_{ij} \rangle\}$
12 **return** $\mathscr{T}_{\text{rev}}$

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 $\mathscr{T}$ 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 set-point, namely, the critical point beyond which navigation is forbidden (lines 16, 19, Algorithm 1). We also assume that robot controllers do not lead robots to

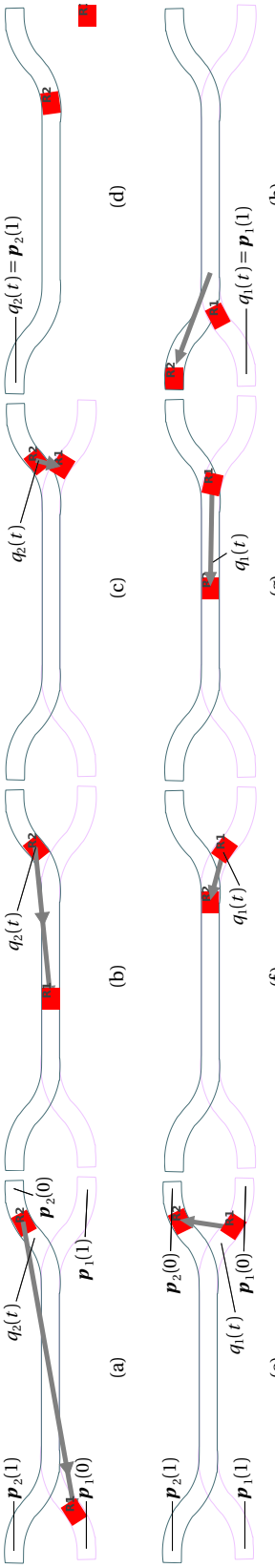


Figure 2: Two examples of robots yielding to each other: (a–d) four moments during navigation in opposing directions; (e–h) four moments during navigation in the same direction. An arrow from i to j indicates that robot i's critical point depends on the progress of robot j, that is, $\mathscr{T}$ contains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

reach poses that lie outside spatial envelopes (which would invalidate Lemma 1).

2.3 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 $C_{ij} \in \mathscr{C}$, the current states s_i and s_j of the involved robots, and the set of precedence constraints $\mathscr{T}$, the robot ordering (k,m) returned by `computeOrdering` is one of the two permutations $\{(i,j),(j,i)\}$, determined as follows:

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

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 (see lines 6, 16, 19 in Algorithm 1). 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 $\mathscr{T}$ is at all times a complete ordering of robots for critical sections $\mathscr{C}$, hence robots are guaranteed not to collide if the temporal profiles they compute when receiving a trajectory update (Algorithm 1, line 16) ensure that the critical point is reached with zero velocity and not passed (Lemma 1). □

2.4 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

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

$$\dot{q}_i^{t+\Delta t} \approx \dot{q}_i^t + \ddot{q}_i^{t+\Delta t}\Delta t, \tag{8}$$

$$q_i^{t+\Delta t} \approx q_i^t + \dot{q}_i^{t+\Delta t}\Delta t, \tag{9}$$

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 $(i <_{C_{ij}} j) \in \mathscr{T}$, 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},

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

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

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 $\mathscr{T}$, 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

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

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 16). 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 $\mathscr{T}$ is at all times a complete ordering of robots for critical sections $\mathscr{C}$, hence robots are guaranteed not to collide if the temporal profiles they compute when receiving a trajectory update (Algorithm 1, line 16) ensure that the critical point is reached with zero velocity and not passed (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

$$h_{\text{dist}}(s_k, s_m, k, m) \equiv \boldsymbol{p}_k(\inf_{\boldsymbol{p}_k} C_{km}) - \boldsymbol{p}_k(q(s_k)), \tag{12}$$

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 $\mathscr{T}$, or that consider the forward models of robots to account for best-, average- and worst-case performance.

2.5 Deadlocks

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

Definition 10. Let $\mathscr{T}$ be a set of precedence constraints over robot paths $\mathscr{P}$ with critical sections $\mathscr{C}$, and let $D_{\mathscr{T}} = (V,E)$ be the dependency graph of the constraints, namely:

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

If $D_{\mathscr{T}}$ contains a cycle $\langle i_1, \ldots, i_m = i_1 \rangle$, then $\mathscr{T}$ 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 $\boldsymbol{p}_{i_j}(q_{i_j}) > \boldsymbol{p}_{i_j}(q'_{i_j})$ for all $j \in [2..m]$.

An unsafe cycle describes the situation in which every robot involved in the cycle requires another robot to reach a pose that will never be reached. Under the assumption that robots are not in critical sections at the beginning and end of their paths ($\boldsymbol{p}_i(\inf_{\boldsymbol{p}_i} C_{ij}) > \boldsymbol{p}_i(0)$ and $\boldsymbol{p}_i(\sup_{\boldsymbol{p}_i} C_{ij}) < \boldsymbol{p}_i(1)$ for all $i \in [1..n]$), this captures all and only the situations in which a deadlock may occur, that is:

Remark 3. If $\mathscr{T}$ 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 $\mathscr{T}$ is deadlock-free.

The simple robot ordering policy described in eq. (6) ensures the complete absence of cycles under the assumption that $|G| \leq 1$ at each iteration (at most one goal is posted per period in Algorithm 1), as this entails that any other robot will have priority over the robot with a new goal, and this ordering will not be changed 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 under the assumption of 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 $\mathscr{T}$ 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 $\boldsymbol{p}_i(\sup_{\boldsymbol{p}_i} C_{ij}) = \boldsymbol{p}_i(1)$ (robot i "parks" inside critical section C_{ij}), allowing robot j to precede robot i may avert the deadlock.[1] Of course, 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.

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 to reach a new goal, consider an obstacle $R_j(q_j)$ for each robot $j \neq i$ in configuration q_j such that $\langle \boldsymbol{p}_j, \boldsymbol{p}_i, q_j, q_i \rangle \in \mathscr{T}$. A similar form of prioritized planning has been described in [16], and is shown to avert deadlocks under the assumption that the infrastructure is "well-formed", a notion that corresponds to a specific case of our earlier assumption that $\boldsymbol{p}_i(\inf_{\boldsymbol{p}_i} C_{ij}) > \boldsymbol{p}_i(0)$ and $\boldsymbol{p}_i(\sup_{\boldsymbol{p}_i} C_{ij}) < \boldsymbol{p}_i(1)$ for every robot i.

2.6 Empirical Validation

The loosely-coupled approach described above has been implemented as an open-source general purpose library, and is available on GitHub [11]. Three properties of the approach have been evaluated, namely:

[1] This strategy is similar to that of adding null-segments in the collision regions of task completion diagrams described in [9].

- Scalability (number of robots and critical sections);
- Ability to account for robot dynamics;
- Realizability with different robot platforms, motion planning and control approaches.

We summarize here the results of these evaluations, while full details will be available in a forthcoming paper (currently under review for ICAPS 2018). We also describe a case-study in which the approach is used with simulated forklifts and the sampling-based motion planning solution described in Deliverable D5.2.

Scalability. To assess scalability, two tests were conducted using a low-fidelity simulation back-end. Paths are pre-planned and loaded from a library. In each test, an increasing number of robot goals were asynchronously posted to the `coordination` algorithm, culminating in 50 active robot goals at one time. The algorithm was run with $T = 0\,\text{s}$, that is, the period length was the time needed to perform all operations in one iteration of the outer loop (lines 2–21, Algorithm 1). The tests reveal that the minimum period length depends on the number of robots that move concurrently, and not on the total number of robots in the fleet. Also, the tests provide evidence of the fact that the approach is suitable for realistically-sized applications (numbering in the tens of robots).

Accounting for robot dynamics. An important feature of the loosely-coupled approach is that, while it can exploit knowledge of robot dynamics to change precedences through critical sections on the fly, it does not depend on this knowledge to determine the timing of communications to the robots. This is a useful feature, as communicating often may not be possible in real situations. An experiment was thus performed to assess how increasing the period T at which the `coordination` loop iterates (which is the period at which communication occurs with the robots) affects the performance of the fleet. Here, performance is measured in terms of average time to completion of robot trajectories. We postulated that average time to completion would increase as the period T increases, due to the fact that the actual dynamic behavior of robots is re-assessed less often, leading to less smooth behavior of the fleet. The principal finding of this experiment is a confirmation of this hypothesis. We also found that robot average time to completion degrades linearly with T.

Use cases with real robots. A number of tests were performed with simulated and real robots. Two tests were performed with two CiTi-truck forklifts in the basement of AASS at Örebro University. In both tests, goals were asynchronously posted which required the robots to reach five destinations along a loop in the environment (see Figure 3). In one experiment, we let the robots complete 20 cycles, for a total duration of 45 minutes. The robots successfully completed all missions, coordinating as appropriate in all critical sections. A video of the full run is available online (https://youtu.be/9Pr5kVvHSxs). In a second experiment, we induced errors in the robots, which led the `coordination` algorithm to change precedence constraints on the fly to account for the fact that one robot was temporarily blocked. A video of this experiment is also available online (https://youtu.be/M1YQLTn_GGo). In both experiments, a lattice-based motion planner [1] was used, all motion planning and control algorithms were run on the robots, and the `coordination` algorithm was run on a separate laptop connected to the fleet via ROS. The implementation of the ROS interface is available as a separate ROS package on GitHub [10].

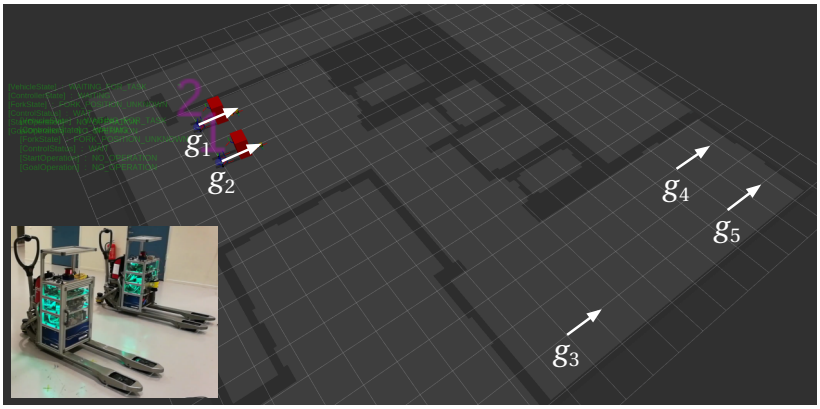


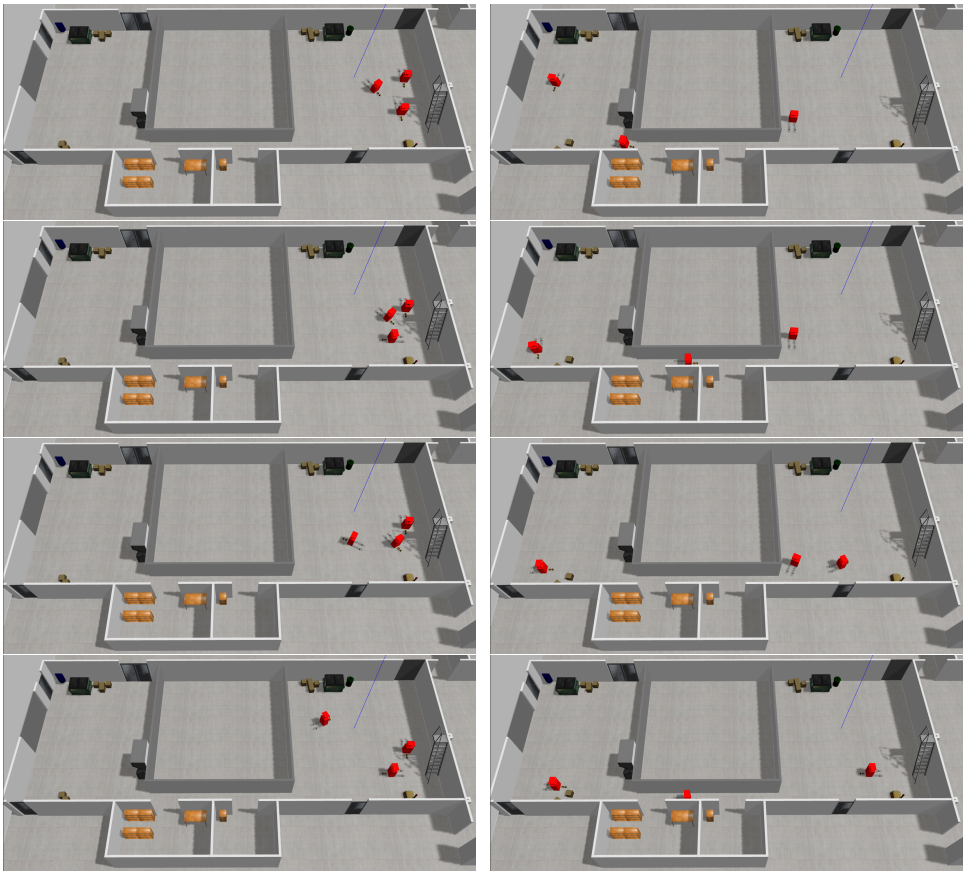
Figure 3: RViz view of the two robots in the basement. Robot 1 is continuously posted goals $\{g_3, g_4, g_2\}$ in sequence, while robot 2 cycles through goals $\{g_3, g_5, g_1\}$. The two robots used in the experiment are shown in the inset.

Figure 4: Gazebo simulator view of the NCFM environment with three robots.

NCFM use case simulation. Finally, a realization of a food production scenario was tested in a simulated environment modeled after the NCFM production facility in Holbeach (at ILIAD partner UoL). Three robots are placed in the environment as shown in see Figure 4, and goals $\{g_1, g_2, g_3, g_4\}$ are continuously posted to all three robots. This leads the coordination algorithm to impose precedences in the areas surrounding the goals, as well as the long corridor connecting the two halves of the environment (see screenshots in Figure 5). The simulation was run for 48 hours to test the reliability of the system, and was tested with both the lattice-based motion planning approach described in [1] and the sampling based approach developed by partner Bosch and described in Deliverable D5.2.

2.7 Strengths and Limitations of the Approach

The approach presented in this section has a number of advantages related to deployability. The method 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. Also, the method does not impose



(a) (b)

Figure 5: Two instances of robot coordination in the NCFM scenario: (a) approaching goal g_1; (b) navigating through the corridor connecting the two halves of the environment.

a particular choice of goal allocation strategy: any strategy can be employed to synthesize goal allocations, and it is assumed (and expected, see Section 5) that goal allocations result from an off-the-shelf optimization process which accounts for scenario-specific objective functions. 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.

The method also has a number of drawbacks. While the use of off-the-shelf optimization methods for goal allocation can be envisaged, it is important to note that neither robot paths nor the inference of precedence constraints account for the metrics used in goal allocation. Recall that all four sub-problems of the overall fleet management problem as defined in Section 1 are intimately connected, and solving one independently of the others will lead to sub-optimal solutions or even incompleteness. This drawback suggests that a more tightly-coupled approach should be studied, in which the goal allocation, motion planning, coordination and control problems are solved in a mutually-aware fashion. An initial study in this direction is described in the next section.

3 Towards a Tightly-Coupled Approach

The motion of a robot can be considered as being the consequence of its sate adhering to constraints that change over time. This view matches well with the fleet management problem statement introduced in Section 1. Each of the four sub-problems of the fleet management problem can be seen as the imposition of additional constraints: solving the goal allocation problem means imposing that each given goal coincides with the final configuration of a robot; coordination means to impose that robot configurations are not overlapping at any point in time; motion planning and control can be seen as the imposition of constraints which restrict motions of robots to kino-dynamically feasible ones. In this section, we begin by providing a general, formal definition of the overall fleet management problem in terms of constraints. We then discuss what it takes to solve a problem formulated in such a way, and propose an initial refinement of this formulation that is amenable to off-the-shelf solvers.

3.1 General Formulation as a Constraint Optimization Problem

In this section, an optimization formulation of the fleet management problem is presented in a general form. The decision variables of this problem are functions which map both state and control of the robot fleet over time $(\mathbf{x}(t), \mathbf{u}(t))$ and a set of discrete variables $\mathbf{d}(.)$ which represent boolean decisions (e.g. robot-goal allocation).

The control space of the multirobot fleet is the R-ary cartesian product, $\mathscr{U}_{\text{fleet}} \subseteq \mathscr{U}_{r_1} \times ... \times \mathscr{U}_{r_R} = \{\mathbf{u}(t) \in \mathbb{R}^{n_u \times R}\}$, of each robot control space, $\mathscr{U}_{r_i} = \{\mathbf{u}_{r_i}(t) \in \mathbb{R}^{n_u}\}$. Similarly, the state space of the multirobot fleet is the R-ary cartesian product , $\mathscr{X}_{\text{fleet}} \subseteq \mathscr{X}_{r_1} \times ... \times \mathscr{X}_{r_R} = \{\mathbf{x}(t) \in \mathbb{R}^{n_x \times R}\}$, of each robot state space, $\mathscr{X}_{r_i} = \{\mathbf{x}_{r_i}(t) \in \mathbb{R}^{n_x}\}$. There are decisions which are inherently discrete/boolean, e.g. "robot r goes to goal g", then the necessity to define a decision space for the multirobot fleet, which is the R-ary cartesian product, $\mathscr{D}_{\text{fleet}} = \mathscr{D}_{r_1} \times ... \times \mathscr{D}_{r_R}$, of each robot decision space defined as $\mathscr{D}_R = \{\mathbf{d}_{r_i}(.) \in \{0,1\}\}$, e.g. $\mathbf{d}_{r_1}(g_3)$ which is a proposition of the form is robot 1 "Assigned to goal 3", for the sake of simplicity the formulation would be utilized in the form d_{r_1,g_3}. N.B. n_u, n_x, R, represent respectively the cardinality of control inputs, robot states and robots.

$$\text{Minimize} \quad L(\mathbf{x}(t),\mathbf{u}(t),t,\mathbf{d}(.)) = \int_{t_0}^{t_f} l(\mathbf{x}(t),\mathbf{u}(t),t,\mathbf{d}(.))dt + \psi(\mathbf{x}(t_f),\mathbf{d}(.)) \quad \text{Cost Function}$$

$$\mathbf{x}(t)\in\mathscr{X}_{\text{fleet}}$$

$$\mathbf{u}(t)\in\mathscr{U}_{\text{fleet}}$$

$$\mathbf{d}(.)\in\mathscr{D}_{\text{fleet}}$$

subject to: $\forall t \in [t_0, t_f]: \; r \in \{r_1, ..., r_R\}: \; \forall t_k \in [t_0, t_f], \; k \in \mathbb{N}$

$f_r(\mathbf{x}_r(t),\mathbf{u}_r(t)) - \dot{\mathbf{x}}_r(t) = \mathbf{0}:$	Dynamic Model Constraints
$B(\mathbf{x}(t_k),\mathbf{d}(.)) = 0$	Boundary/Point Constraints
$P(\mathbf{x}(t),\mathbf{e}(t)) \leq 0$	$\mathscr{C}_{\text{free}}$Constraints
$\underline{u} \leq \mathbf{u}_r(t) \leq \bar{u}$	Control Constraints

Where

L	: Continuous Time Cost Function
l	: Lagrangian Term of L
ψ	: Endpoint Term of L
f_r	: Robot r Dynamics
B	: Boundary Constraints
P	: Path Constraints
e	: Environment State
$\bar{u}$	: Upper Bound on the Control Input
$\underline{u}$	: Lower Bound on the Control Input
t_0, t_f	: Starting and final time respectively

The formulation captures all four sub-problems stated in Section 1:

- The goal allocation problem is that of assigning decision variables $\mathbf{y}_{r,g} \in \mathscr{D}_r$: robot r is assigned to reach goal g iff $\mathbf{y}_{r,g} = 1$. Boundary/point constraints impose that the final poses of robots along their paths correspond to the given goals, and that each goal is reached by exactly one robot.
- The motion planning problem is captured by path constraints, which restrict decision variables $\mathbf{x}_r(t)$ (representing the configurations of a robot over time) to be in an obstacle-free part of the environment ($\mathbf{e}(t)$).
- The coordination problem is also captured by path constraints, which can be assumed to incorporate minimum distances between robots at all time points ($P(t)$).
- The control problem is modeled via constraints on decision variables $\mathbf{u}(t)$. These include the dynamic models f_r of the robots, and bounds $[\underline{u}, \bar{u}]$ on the control inputs.

We instantiate the above formulation in a simple scenario (see Figure 6) in which three robots are assigned two goals in an obstacle-free environment.

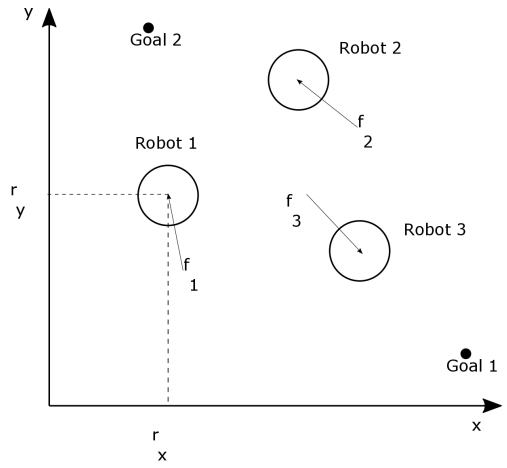


Figure 6: A simple example scenario, where the configuration of a robot is defined by its position (r_x, r_y), the control input is a force (f_1, f_2, f_3) applied in the mass center of the robot, and there are three robots and two goals.

Objective function. We start by choosing the objective function

$$L(\mathbf{x}(t),\mathbf{u}(t)) = \sum_{r=1}^{\mathbf{R}} \int_{t_0}^{t_f} \mathbf{x}_r(t)^T Q\, \mathbf{x}_r(t) + \mathbf{u}_r(t)^T \mathbf{R}\, \mathbf{x}_r(t) d\,t \tag{13}$$

$$\begin{aligned} \Leftrightarrow L(\mathbf{x}(t),\mathbf{u}(t)) = & \int_{t_0}^{t_f} \mathbf{x}_{r_1}(t)^T \mathbf{Q}\, \mathbf{x}_{r_1}(t) + \mathbf{u}_{r_1}(t)^T \mathbf{R}\, \mathbf{u}_{r_1}(t) d\,t + \\ & \int_{t_0}^{t_f} \mathbf{x}_{r_2}(t)^T \mathbf{Q}\, \mathbf{x}_{r_2}(t) + \mathbf{u}_{r_2}(t)^T \mathbf{R}\, \mathbf{u}_{r_2}(t) d\,t + \\ & \int_{t_0}^{t_f} \mathbf{x}_{r_3}(t)^T \mathbf{Q}\, \mathbf{x}_{r_3}(t) + \mathbf{u}_{r_3}(t)^T \mathbf{R}\, \mathbf{u}_{r_3}(t) d\,t \end{aligned}$$

which minimizes control energy spent by the robots.

Dynamic model. The dynamic model of the robots can be defined with ordinary differential equations.

$$f(\mathbf{x}(t),\mathbf{u}(t)) = \dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{14}$$

$$\Leftrightarrow \underbrace{\begin{bmatrix} \dot{\mathbf{x}}_{r_1}(t) \\ \dot{\mathbf{x}}_{r_2}(t) \\ \dot{\mathbf{x}}_{r_3}(t) \end{bmatrix}}_{\dot{\mathbf{x}}(t)} = \underbrace{\begin{bmatrix} \mathbf{A}_{r_1} & 0 & 0 \\ 0 & \mathbf{A}_{r_2} & 0 \\ 0 & 0 & \mathbf{A}_{r_3} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} \mathbf{x}_{r_1}(t) \\ \mathbf{x}_{r_2}(t) \\ \mathbf{x}_{r_3}(t) \end{bmatrix}}_{\mathbf{x}(t)} + \underbrace{\begin{bmatrix} \mathbf{B}_{r_1} & 0 & 0 \\ 0 & \mathbf{B}_{r_2} & 0 \\ 0 & 0 & \mathbf{B}_{r_3} \end{bmatrix}}_{\mathbf{B}} \underbrace{\begin{bmatrix} \mathbf{u}_{r_1}(t) \\ \mathbf{u}_{r_2}(t) \\ \mathbf{u}_{r_3}(t) \end{bmatrix}}_{\mathbf{u}(t)}$$

And

$$\underbrace{\begin{bmatrix} \dot{\mathbf{q}}_r \\ \ddot{\mathbf{q}}_r \end{bmatrix}}_{\dot{\mathbf{x}}_r(t)} = \mathbf{A}_r \underbrace{\begin{bmatrix} \mathbf{q}_r \\ \dot{\mathbf{q}}_r \end{bmatrix}}_{\mathbf{x}_r(t)} + \mathbf{B}_r^T \underbrace{\mathbf{f}_r}_{\mathbf{u}_r(t)} : r = \{r_1, r_2, r_3\}$$

where

$\mathbf{A}_r$: State Matrix of Robot r
$\mathbf{B}_r$: Control Matrix of Robot r
$\mathbf{q}_r$: Generalized Coordinate System, where $\mathbf{q}_r = [r_x, r_y]^T$
$\mathbf{f}_r$: Force Vector applied to robot r, where $\mathbf{f}_r = [f_{r_x}, f_{r_y}]^T$

In the particular case in which all robots follow the same model, the control and state matrices are equal: $A_{r_1} = A_{r_2} = A_{r_3} = A$ and $B_{r_1} = B_{r_2} = B_{r_3} = B$.

If the system dynamics are linear, then the model can be discretized as follows: $X(k+1) = AX(k) + BU(k) : k = 1, ..., N$ and $N \times \tau = T$, where T is the total trajectory time, that is, τ is a time step.

Boundary/Point constraints. In the model depicted in eq. (14) by knowing the initial state $\mathbf{x}(0)$ and the control space $\mathbf{u}(t) : \forall\, t \in [t_0, t_f]$ it is possible to reach all the states, if we assume the system is controllable, although such information about the system isn't known *a priori* . Nevertheless, one could desire to define a final/goal state, checkpoints, or have a periodic sequence where the robot has a certain goal and then has to return to the initial position. All of these additional requirements can be defined by point constraints. In this example, it will be defined initial state constraints and goals constraints. N.B. there is more robots then goals, as a result we are also formulating a goal assignment problem.

$$B(\mathbf{x}_r(t_k), \mathbf{y}) = \begin{cases} B_I(\mathbf{x}_r(0)) \text{initial value constraints} \\ B_F(\mathbf{x}_r(t_f), y_{r,g}) \text{ goal constraints} \end{cases} \tag{15}$$

Where $\mathbf{y} = \mathbf{d}_r(g) \in \mathscr{D}_{free}, \forall\, r \in \{r_1, r_2, r_3\}$: $\forall\, g \in \{r_1, r_2\}$ accordingly

$$B_I(\mathbf{x}_r(0)) = \begin{cases} x_{r_1}(0) - x_{r_{1_0}} = 0 \\ x_{r_2}(0) - x_{r_{2_0}} = 0 \\ x_{r_3}(0) - x_{r_{3_0}} = 0 \end{cases}$$

$$B_F(\mathbf{x}_r(t_f), \mathbf{y}_{r,g}) = \begin{cases} \|\mathbf{x}_{r_1}(t_f) - \mathbf{x}_{g_1}\|_2 - \|\mathbf{x}_{r_1}(t_f) - \mathbf{x}_{g_1}\|_2 \cdot y_{r_1,g_1} \geqslant 0 \\ \|\mathbf{x}_{r_1}(t_f) - \mathbf{x}_{g_2}\|_2 - \|\mathbf{x}_{r_1}(t_f) - \mathbf{x}_{g_2}\|_2 \cdot y_{r_1,g_2} \geqslant 0 \\ \|\mathbf{x}_{r_2}(t_f) - \mathbf{x}_{g_1}\|_2 - \|\mathbf{x}_{r_2}(t_f) - \mathbf{x}_{g_1}\|_2 \cdot y_{r_2,g_1} \geqslant 0 \\ \|\mathbf{x}_{r_2}(t_f) - \mathbf{x}_{g_2}\|_2 - \|\mathbf{x}_{r_2}(t_f) - \mathbf{x}_{g_2}\|_2 \cdot y_{r_2,g_2} \geqslant 0 \\ \|\mathbf{x}_{r_3}(t_f) - \mathbf{x}_{g_1}\|_2 - \|\mathbf{x}_{r_3}(t_f) - \mathbf{x}_{g_1}\|_2 \cdot y_{r_3,g_1} \geqslant 0 \\ \|\mathbf{x}_{r_3}(t_f) - \mathbf{x}_{g_2}\|_2 - \|\mathbf{x}_{r_3}(t_f) - \mathbf{x}_{g_2}\|_2 \cdot y_{r_3,g_2} \geqslant 0 \\ y_{r_1,g_1} + y_{r_2,g_1} + y_{r_3,g_1} = 2 \;;\; y_{r_1,g_2} + y_{r_2,g_2} + y_{r_3,g_2} = 2 \\ y_{r_1,g_1} + y_{r_1,g_2} \geqslant 1 \;;\; y_{r_2,g_1} + y_{r_2,g_2} \geqslant 1 \;;\; y_{r_3,g_1} + y_{r_3,g_2} \geqslant 1 \end{cases} \tag{16}$$

$x_{r_0} \in \mathscr{X}_r$: Initial position of robot r
$x_g \in \mathscr{X}_g$: goal configuration $\mathscr{X}_g \subseteq \mathscr{X}_r$
$\|\cdot\|_2$: L2-norm
$y_{r,g} \in \mathscr{D}_r$: Proposition stating robot r is assigned or not to goal g. Note: $y_{r,g} = y_r(g)$ can be utilized interchangeably

The inequalities in expression for $B_F(\mathbf{x}(t_f), y_{r,g})$, of which there is one for every robot-goal combination, are in the form

$$\|\mathbf{x}_r(t_f) - \mathbf{x}_g\|_2 - \|\mathbf{x}_r(t_f) - \mathbf{x}_g\|_2 \cdot y_{r,g} \leqslant 0 \tag{17}$$

They impose that decision variable $y_{r,g} = 0$ iff robot r is assigned to goal g. Note that if $y_{r,g} = 1$, the expression evaluates to zero and the constraint has no effect. Note also that a

robot may be assigned a goal or not. These requirements are imposed by the sums in the expression for $B_F(\mathbf{x}(t_f), y_{r,g})$, which are in the form

$$\begin{aligned} \textstyle\sum_{i=1}^{G} y_{r,g} &= R-1 \quad [g = g_1, ..., g_G] \\ \textstyle\sum_{j=1}^{R} y_{r,g} &\geqslant G-1 \quad [r = r_1, ..., r_R] \end{aligned} \tag{18}$$

where:

G : Number of Goals

Path constraints. Path constraints are used to impose requirements on traversable space. In this example, we assume a workspace without obstacles, hence the free space consists of the entire joint configuration space of all robots, excluding those configurations in which robots collide with one another. Such constraints can be represented as follows:

$$P(\mathbf{x}(t), \mathbf{e}(t)) = P(\mathbf{x}(t)) = \begin{bmatrix} P_{r_1,r_2}(x_{r_1}(t), x_{r_2}(t)) \\ P_{r_1,r_3}(x_{r_1}(t), x_{r_3}(t)) \\ P_{r_2,r_3}(x_{r_2}(t), x_{r_3}(t)) \end{bmatrix} \tag{19}$$

Where

$$P_{r_i,r_j}(x_{r_i}(t), x_{r_j}(t)) = \|x_{r_i}(t) - x_{r_j}(t)\|_2 - d_{r_i,r_j} \geq 0$$

P_{r_i,r_j} : Collision free path constraint between robot r_i and robot r_j
d_{r_i,r_j} : Safety distance between robot r_i and robot r_j

Clearly, this representation of the problem is unwieldy, if only because the total amount of path constraints is combinatorial in the number of robots: extrapolating to R robots , there are a total of ${}^{R}C_2 = \frac{R!}{2!(R-2)!}$ path constraints. Furthermore, the formulation assumes holonomic point robots, and does not include obstacles. Even under such simplified assumptions, finding a solution is computationally expensive. Moreover, state and control space variables have continuous domains, i.e., $\mathbf{u}_r(t), \mathbf{x}_r(t)$. Note, however, that each $y_{r,g}$ assumes values in $\{0,1\}$. This leads to a non-convex solution space. Any solver relying on a Newton method will thus be trapped in a convex subspace of the solution space, therefore entirely missing solutions that require different assignments of the discrete variables. This suggests a different approach to modeling (and solving) our problem, which we outline in the next section.

3.2 A Two-Phase Approach

Given that goal allocation variables $y_{r,g} \in \mathscr{D}_{\mathrm{r}}$ are discrete/integer variables, we can formulate an abstraction of our problem as a Mixed Integer Quadratic Problem (MIQP), which can be solved efficiently if the cost function is quadratic and the constraints are either linear or a second order cone. We therefore propose a formulation of the fleet management problem consisting of a cost function, a holonomic robot model with liner differential equations, point-shaped holonomic robots, and a discretization of the obstacle-free configuration space into overlapping convex polygons. The goal assignment problem is modeled via integer decision variables. The solution of the resulting MIQP with quadratic constraints prescribes (1) an allocation of goals to robots ($y_{r,g}$), (2) the spatial constraints (convex polygons) within which each robot configuration should lie, and (3) a control input sequence for each robot ($\mathbf{u}_{r_x}$,$\mathbf{u}_{r_y}$) which leads the robot from its initial state to a goal location. The solution of the MIQP can then be used as initial guess for solving a Non-Linear Problem (NLP) which accounts for the requirements that were relaxed in the MIQP formulation.

3.2.1 Initial Solution of a Relaxed Problem (MIQP)

The MIQP formulation is provided below.

$$\text{Minimize} \quad L(\mathbf{u}(.)) = \sum_{r=1}^{R}\sum_{k=1}^{N} \mathbf{u}_r(k)\mathbf{R}\mathbf{u}_r^T(k) \qquad \text{Cost Function}$$

$\mathbf{u}_r(k) \in \mathscr{U}_r$, $\forall k \in [1,...,N] : \forall r = [1,...,R]$

$y_{g,r} \in \mathscr{D}_{\text{fleet}}$, $\forall r = [1,...,R] : \forall g = [1,...,G]$

$\mathbf{c}_{r,p}(k) \in \mathscr{D}_{\text{fleet}}$, $\forall r = [1,...,R] : \forall g = [1,...,G] : \forall k = [1,...,N]$

subject to: $\forall r \in [1,...,R] : \forall k \in [1,...,N] : \forall p \in [1,...,P] : \forall g = [1,...,G]$

$$\mathbf{x}_r(0) = [\mathbf{q}_r(0)\ \ \dot{\mathbf{q}}_r(0)]^T = \mathbf{x}_{r_0} \qquad \text{Initial Condition}$$

$$\mathbf{x}_r(k) = f(\mathbf{x}_r(k-1), \mathbf{u}_r(k)) = \mathbf{A}\mathbf{x}_r(k-1) + \mathbf{B}\mathbf{u}_r(k) \qquad \text{Dynamic Model}$$

$$(\mathbf{x}_g - \mathbf{x}_r(N))y_{g,r} = \mathbf{0}^T \qquad \text{Goal Allocation}$$

$$\sum_{g=1}^{G} y_{g,r} = 1 \qquad \text{Robot has 1 Goal}$$

$$\sum_{r=1}^{R} y_{g,r} = 1 \qquad \text{Goal has 1 Robot}$$

$$\mathbf{A}_p{}^T\mathbf{q}_r(k) + \mathbf{b} - m\mathbf{1}^T(1 - \mathbf{c}_{r,p}(k)) \leq \mathbf{0}^T : m\mathbf{1}^T \ll \mathbf{A}_p{}^T\mathbf{q}_r(k) + \mathbf{b} \qquad \mathbb{C}_{free} \text{ Constraints}$$

$$1 \leq \sum_{p=1}^{P} \mathbf{c}_{r,p}(k) \leq P \qquad \text{Robot is in } \mathscr{C}_{\text{free}}$$

$$\sum_{p=1}^{P} |\mathbf{c}_{r,p}(k) - \mathbf{c}_{r,p}(k+1)| - \sum_{p=1}^{P} \mathbf{c}_{r,p}(k) \leq 0 \qquad \text{Corner Cutting}$$

$$\underline{u} \leq \mathbf{u}_r(k) \leq \bar{u}, \qquad \text{Actuator Constraints}$$

Example case

$$\mathbf{u}_r(k) = \begin{bmatrix} \mathbf{f}_{r_x}(k) \\ \mathbf{f}_{r_y}(k) \end{bmatrix} \ ; \ \mathbf{x}_r(k) = \begin{bmatrix} \mathbf{q}_r(k) \\ \dot{\mathbf{q}}_r(k) \end{bmatrix} \mathbf{R} = \begin{bmatrix} w_{u_x} & 0 \\ 0 & w_{u_y} \end{bmatrix} \tag{20}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \ ; \ \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{M} & 0 \\ 0 & \frac{1}{M} \end{bmatrix} \ ; \ \mathbf{x}_g = \begin{bmatrix} \mathbf{q}_g \\ \dot{\mathbf{q}}_g \end{bmatrix} \tag{21}$$

$$\mathbf{A}_p = \begin{bmatrix} \mathbf{a}_{p_1} & \mathbf{a}_{p_2} & \dots & \mathbf{a}_{p_{HP}} \end{bmatrix} \; ; \; \mathbf{b} = \begin{bmatrix} b_{p_1} \\ b_{p_2} \\ \vdots \\ b_{p_{HP}} \end{bmatrix} \tag{22}$$

$\mathbf{x}_g \in \mathcal{X}_g$: Goal Configuration $\mathcal{X}_g = \mathcal{X}_r$
$\mathbf{q}_r$: Generalized Coordinate System, where $\mathbf{q}_r = [r_x, r_y]^T$
$\mathbf{f}_r$: Force Vector applied to robot r, where $\mathbf{f}_r = [f_{r_x}, f_{r_y}]^T$
$\mathbf{R}$: Control input weighted matrix
w_{u_x} : Longitudinal control input weight
w_{u_y} : Lateral control input weight
M : Mass of the robot
$y_{g,r}$: Goal-Robot allocation variables - robot r assigned to goal g
$\mathbf{c}_{r,p}(k)$: Robot-state-polygon allocation variables - robot r at state k is in polygon p
$\mathbf{a}_{p_i}$: Normal vector of the halfspace i of polygon p
m : Large arbitrary positive value
HP : Number of Polygon half-planes
R : Number of robots where r is the robot index
N : Number of robot states where k is the robot state index
G : Number of goals where g is the goal index
P : Number of polygons where p is the polygon index

The cost function minimizes control energy, as is common in robotic optimization problems. The cost function is a sum-of-squares. Since it is a convex function, one can apply many off-the-shelf algorithms.

The initial condition sets all the robots to their initial state. The dynamic model is composed of linear differential equations, and is also convex.

The goal allocation condition states that the final state of some robots have to be the same as the goal state; which robots are assigned to each goal is not assumed to be *a priori* knowledge, rather it is determined as a result of two further conditions: robots can only have one goal, and each goal can only be fulfilled by one robot.

The configuration free space is given as a disjunction of polytopes defined as a conjunction of half-planes. In order for the robot to not collide with static obstacles in the environment, each state has to lie within at least one of the polytopes. These constraints impose the safety of individual states, but not of the transitions between them. Hence, a constraint is added imposing that two consecutive states have to lie within the same polytope.

Finally, torques on motors are limited, and those limitations are accounted for as physical constraints on the control inputs.

The formulation has a quadratic cost function and linear constraints, and as a result, the problem as defined above is convex. In such problems, local minima are also global minima, and only first-order optimality conditions have to be met.

The formulation above uses several patterns commonly used in Mixed-Integer Programming. The constraints stating that robot states must lie within a polygon uses one such pattern, namely, a “Big-M constraint” was utilized. This type of constraints has the ability to “activate” or “deactivate” certain constraints, allowing to construct a formulation with disjunctive statements. As a result, depending on whether the variable $\mathbf{c}_{r,p}(k)$ is assigned value 1 or 0, the constraint is activated or deactivated, respectively. Another such pattern is used in the constraint imposing that two consecutive states have to be inside the same polygon. Here, the absolute value function is substituted by linear constraints:

$$
\begin{aligned}
& y = |\mathbf{c}_{r,p}(k) - \mathbf{c}_{r,p}(k-1)| : c_{(.)} \in \{0,1\} \\
& \mathbf{c}_{r,p}(k) - \mathbf{c}_{r,p}(k-1) - (P_+ - P_-) = 0 \\
& L_y = -1 \ , U_y = 1 \\
& P_+ - U\, P_{+||-} \leq 0 \\
& P_- - |L|(1 - P_{+||-}) \leq 0 \\
& P_{+||-} \in \{0,1\}
\end{aligned}
$$

P_+ : Auxiliary variable which is equal to y if $\mathbf{c}_{r,p}(k) - \mathbf{c}_{r,p}(k-1)$ is positive
P_- : Auxiliary variable which is equal to y if $\mathbf{c}_{r,p}(k) - \mathbf{c}_{r,p}(k-1)$ is negative
L_y : Lower bound on $\mathbf{c}_{r,p}(k) - \mathbf{c}_{r,p}(k-1)$
U_y : Upper bound on $\mathbf{c}_{r,p}(k) - \mathbf{c}_{r,p}(k-1)$
$P_{+||-}$: Auxiliary variable which sets either P_+ or P_- to zero or U and L otherwise respectively

These auxiliary variables and constraints allow to rewrite the absolute value formula as follows:

$$|\mathbf{c}_{r,p}(k) - \mathbf{c}_{r,p}(k-1)| = P_+ + P_-$$

In this formulation, the robot is considered to be a point defined by the Cartesian coordinates of its geometric center $(\mathbf{x}_r(k), \mathbf{y}_r(k))$. Other representations for the robot can be considered, e.g., circle robots, without forfeiting convexity.

The problem formulation above is a relaxation of the overall fleet management problem. Specifically, the relaxed elements of the problem are the robot geometry, the dynamic models of robots, and the fact that robot-robot collisions are ignored. As a result, the solution of the relaxed problem is not an executable control sequence, rather its intended use is as an initial guess for solving the overall problem. The solution, in other words, is used to *inform* a subsequent Non-Linear Problem (NLP) solver.

By considering robots as points, one is assuming that robots can pass through narrow passages, which of course may not be possible when the real geometry is considered. Furthermore, since the optimal control is formulated in a manner that minimizes control energy, robots will tend to navigate close to obstacles, e.g., cutting corners.

The dynamic model of the robot is a composition of linear first order ODEs resulting in an holonomic model and allowing sideways motions, which would not be possible if considering a "car-like" dynamic model. As a result, the solution path of each robot will be a conjunction of straight line segments from the initial to the final robot configuration.

The solution of the MIQP is a combined sequence of control inputs for each robot, finding both the path and temporal profile for each robot subject to the stated constraints. The solution can be considered as a "lower bound" solution for the overall fleet. Assigning for each state of each robot $(\mathbf{x}_r(k))$ a specific polygon, which is a result of solving the discrete variables $(\mathbf{c}_{r,p}(k) \in \{0,1\})$, is one of the interesting features of the MIQP formulation. The solution of the $\mathbf{c}_{(.)}$ variables can be understood as the sequence of polygons the robots must traverse from the initial to the final configuration while considering all the stated constraints. This goes beyond being a geometric solution: knowing that robot r_1 is not on the same polygon as robot r_2 at time t allows to drop collision constraints between these two robots at time t. On the other hand, if both robots are in the same polygon at time t, then collision constraints should be considered. In this case, two possibles to tackle a possible collision can be foreseen: both robots execute evasion maneuvers in the same polygon so as to avoid each other, or one robot transitions to an adjacent polygon. In this

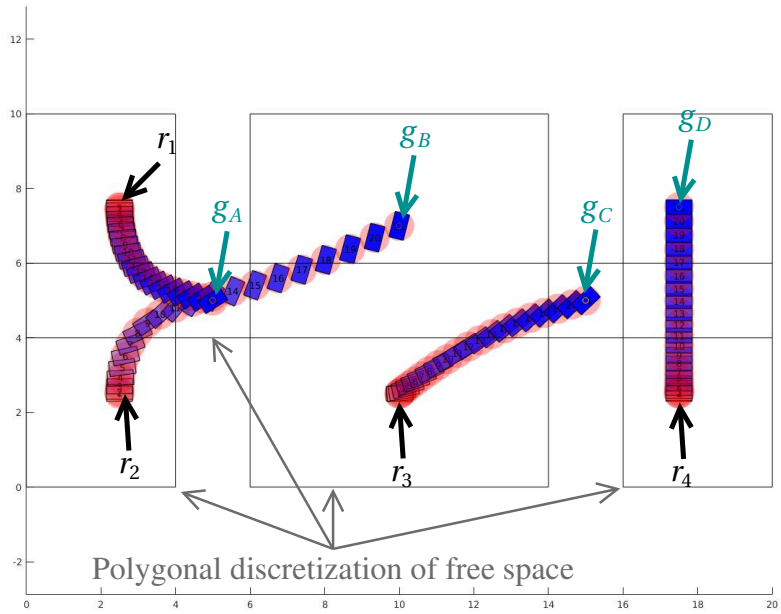


Figure 7: Solution of an example MIQP for four robots and four goals.

sense, the solution of the MIQP carries with it potentially useful information for solving the NLP.

Figure 7 shows a solution of a MIQP formulation of a fleet management problem with four robots and four goals. As shown, the goals assigned to robots r_1, r_2, r_3, r_4 are, respectively, g_A, g_B, g_C, g_D. The trajectories leading to these goals are computed assuming point robots and holonomic kinematics, and no provision is made to avoid collisions between r_1 and r_2.

3.2.2 Solution Refinement (NLP)

Given a solution to the relaxed problem as shown above, we now turn our attention to imposing the remaining constraints (robot-robot collision avoidance, kinematics, robot geometry). Robot geometry is approximated via a conjunction of circles centered along the center-line of the robot footprint. Robot-robot collision constraints are considered pair-wise, and the dynamic model of robots is a combination of nonlinear first-order ordinary differential equations. Additionally, the configuration space of each robot is enhanced by the heading and velocity tangent to the robot's path. The solution computed by the MIQP solver is utilized to pre-assign robots to goals, as well as to determine which polygons a robot should be in at each time step. The control inputs computed in the previous phase are also used as a plausible initial guess for the NLP solver. Cost function, initial configuration of the robot fleet, and physical limitations are maintained unaltered.

Minimize $L(\mathbf{u}(.)) = \sum_{r=1}^{R}\sum_{k=1}^{N} \mathbf{u}_r(k)\mathbf{R}\mathbf{u}_r^T(k)$ Cost Function

$\mathbf{u}_r(k)\in\mathscr{U}_r$, $\forall k\in[1,...,N]:\forall r=[1,...,R]$

subject to: $\forall r\in[1,...,R]:\forall k\in[1,...,N]:\forall p\in[1,...,P]:\forall c\in[1,...,C]$

$\mathbf{x}_r(0) = [\mathbf{r}_x(0)\ \mathbf{r}_y(0)\ \mathbf{r}_\theta(0)\ \mathbf{r}_v(0)]^T$ Initial Condition

$\mathbf{x}_r(k) = f(\mathbf{x}_r(k-1), \mathbf{u}_r(k))$ Dynamic Model

$(\mathbf{x}_g - \mathbf{x}_r(N)) = \mathbf{0}$ $: y_{r,g}=1$ Goal Allocation

$\mathbf{A}_p{}^T \mathbf{q}_r^c(k) + \mathbf{b} - m\mathbf{1}^T\,(1-\mathbf{c}_{r,p}(k)\,) \le \mathbf{0}^T$ $: \mathbf{c}_{r,p}(k)=1$ $\mathbb{C}_{free}$ Constraints

$\|\mathbf{x}_{r_i}^{c_i}(k) - \mathbf{x}_{r_j}^{c_j}(k)\|_2 - \mathrm{ra}_{r_i}^{c_i} - \mathrm{ra}_{r_j}^{c_j}$ $: (j>i)$ Collision Constraints

$\underline{u} \le \mathbf{u}_r(k) \le \bar{u}$ Physical Constraints

Where

$$\mathbf{u}_r(k) = \begin{bmatrix} \mathbf{a}_r(k) \\ \mathscr{K}_r(k) \end{bmatrix};\ \mathbf{R} = \begin{bmatrix} w_{u_a} & 0 \\ 0 & w_{u_\mathscr{K}} \end{bmatrix} \tag{23}$$

$$f(\mathbf{X}_r(k-1), \mathbf{U}_r(k)) = \begin{bmatrix} \mathbf{V}_r(k-1)\ \cos(\theta_r(k-1)) \\ \mathbf{V}_r(k-1)\ \sin(\theta_r(k-1)) \\ \mathscr{K}_r(k)\ \mathbf{V}_r(k-1) \\ \mathbf{a}_r(k) \end{bmatrix};\ \tag{24}$$

$$\bar{u} = \begin{bmatrix} 1 \\ \frac{\tan(\phi_{max})}{l}; \end{bmatrix};\ \underline{u} = \begin{bmatrix} -1 \\ -\frac{\tan(\phi_{max})}{l}; \end{bmatrix};\ \tag{25}$$

- $\mathbf{r}_\theta$: Robot r heading
- $\mathbf{r}_v$: Robot r velocity tangent to the path
- $\mathbf{q}_c^r$: Circle c of robot r Generalized Coordinate System $\mathbf{q}_c^r = [\mathbf{r}_x^c, \mathbf{r}_y^c]$
- w_{u_a} : Acceleration control input weight
- $w_{u_\mathscr{K}}$: Curvature control input weight
- ra_c^r : Radius of cirle c of robot r
- ϕ_{max} : Maximum steering angle
- l : Robot length
- C : Number of circles encapsulating each robot
- c : Circle index

Figure 8 shows the result of solving the NLP formulated above for the same example described earlier. The solution of the MIQP shown in Figure 7 was used as an initial guess for the NLP, yielding a refined set of robot trajectories which adhere to the full set of

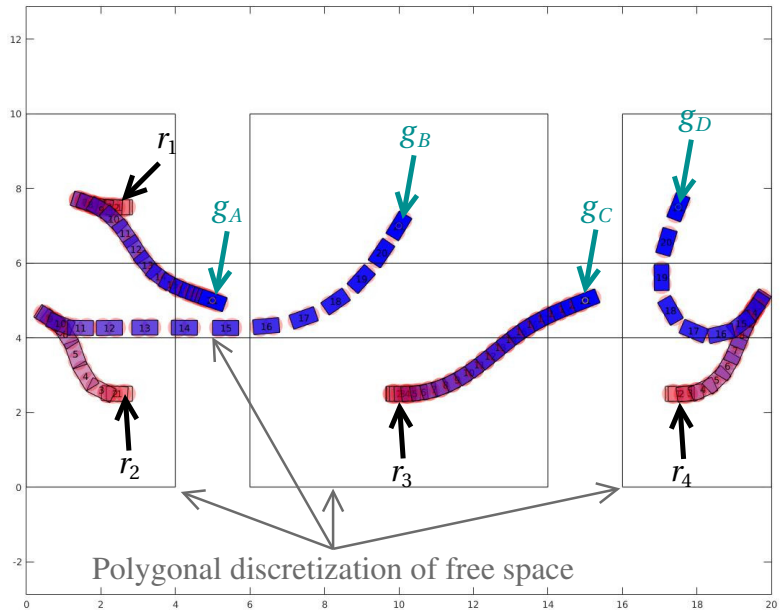


Figure 8: Solution of the NLP, constituting a refinement of the MIQP solution shown in Figure 7.

constraints imposed in the NLP. Specifically, the trajectories now adhere to the kinematic constraints of a car-like robot, and collision between robots r_1 and r_2 are avoided via spatial separation.

If the horizontal polygon spanning the three other polygons in the environment were more narrow, a solution in which r_1 yields to r_2 would be necessary. Figure 9 shows the solutions to the MIQP and NLP in this situation. As shown, collision between r_1 and r_2 is avoided via temporal separation: r_2 occupies the common area in the interval of time $[11, 15]$, while r_1 only reaches the common area after this interval. This constraint is enforced in both the MIQP and NLP solutions, resulting in collision-free motions of the two robots.

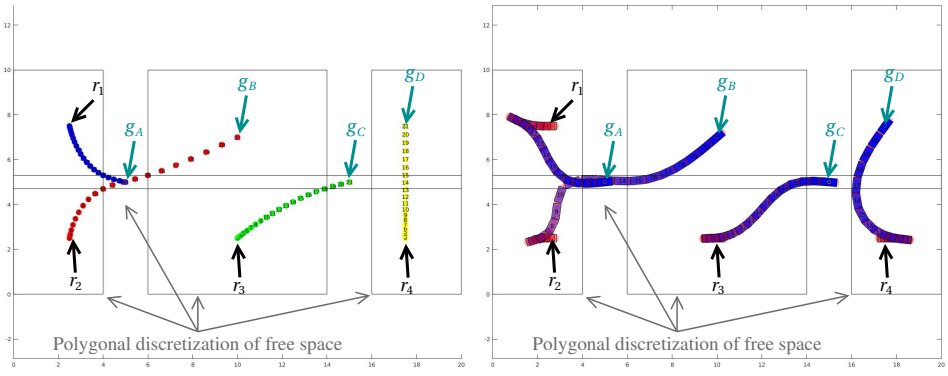


Figure 9: Another solution of the MIQP (left) and NLP problems (right), showing temporal (as opposed to spatial) adjustment to avoid the collision of robots r_1 and r_2.

The two examples we have illustrated show that the tightly-coupled approach possesses a higher level of sophistication than the loosely-coupled approach described in

Section 2. The latter is only capable of performing temporal separation (adjusting the temporal profiles of trajectories), whereas the tightly-coupled approach can seamlessly blend spatial and temporal adjustment.

4 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 [8]. The use of these solutions for automation is therefore limited to environments that can be engineered to suit these assumptions, e.g., warehouses [18].

Graph-based search methods have been employed with some success for coordinating robots. Here, the environment is represented as a graph, whose connectivity represents the possible motions of robots. These methods are often built for maze-like, congested environments, and are not designed for online readjustment [17, 14]. They also make several assumptions: all robots start their missions at the same time, and trajectories are equated to paths through the graph, which is piece-wise continuous. The latter assumption requires robots to stop at each traversed node of the graph to ensure dynamic feasibility. It is shown in [13] and [5] how the resulting collision-free graph paths can be transformed to kinodynamically feasible trajectories via a post-processing step.

A variety of approaches for multi-robot motion planning can be found in the Robotics literature. Most account for robot-robot collisions in via a joint configuration space derived from the Cartesian product of the configuration spaces of all robots in the fleet [7]. However, this approach is computationally expensive for large fleets, and not adequate for online use. Some address the coordination problem in coordination space [9], 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 in [15]. The approach, however, assumes holonomic, disc-shaped robots, and requires all robot paths to be known in order to compute the coordination space (whereas in the loosely-coupled approach described in Section 2, 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 [2, 4]. However, solutions scale badly to a large fleets, due to the complexity of the optimization problem. In [6], an online coordination problem for Multiple Micro Aerial Vehicles (MAVs) is formulated 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, the speed profiles of robots along specified paths can be adjusted to avoid robot-robot collisions [12]. Similarly to the tightly-coupled approach presented in Section 3, 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. A similar approach has been proposed [3] 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.

5 Conclusions and Future Work

In this deliverable we have defined in general terms the fleet management problem that is the focus of ILIAD. We have proposed two solutions to this problem, one relying on a loose coupling of solvers, the other based on a holistic formulation of the fleet management problem as a constrained optimization problem. The loosely-coupled approach is modular, that is, it does not rely on specific choices of goal allocation, motion planning and control methods. Also, it is designed to work in an online setting, where goals are posted while robots are in motion. The approach has been validated formally and experimentally, and tested with simulated and real robots. The loosely-coupled approach is fully implemented as a cross-platform software library [11], and interfaces with the `navigation_oru` stack and the full ILIAD code-base are also available [10].

As discussed, the loosely-coupled approach has a number of drawbacks: neither robot paths nor the inference of precedence constraints account for the metrics used in goal allocation, and there is no provision for the fact that the sub-problems of the overall fleet management problem are intimately connected. These drawbacks are overcome in the tightly-coupled approach, where the goal allocation, motion planning, coordination and control problems are solved in a mutually-aware fashion. This approach allows, as shown in a simple example, the "continuous blending" of goal allocation, coordination, and motion planning solutions.

Future work will further explore applicability and tradeoffs of the two approaches outlined here. Also, for the loosely-coupled approach, we will concentrate on developing the ability to re-plan motions and goal assignments on the fly. These methods will be in turn used to provide deadlock avoidance strategies. For the tightly-coupled approach, we will concentrate on closing the loop between the two solving phases (MIQP and NLP), and begin to address the problems of online goal posting and scalability.

References

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

[2] Federico Augugliaro, Angela P. Schoellig, and Raffaello D'Andrea. 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)*, 2012.

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

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

[5] Wolfgang Hönig, T. K. Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Multi-agent path finding with kinematic constraints. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2016.

[6] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto. 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)*, 2017.

[7] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006.

[8] Hang Ma, Jiaoyang Li, T.K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proc. of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 837–845, 2017.

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

[10] Federico Pecora. *The* `coordination_oru_ros` *Multi-Robot Coordination Package*, 2017. `http://github.com/FedericoPecora/coordination_oru_ros`.

[11] Federico Pecora. *An Online Multi-Robot Coordination Algorithm based on Trajectory Envelopes*, 2017. `http://github.com/FedericoPecora/coordination_oru`.

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

[13] James A. Preiss, Wolfgang Hönig, Nora Ayanian, and Gaurav S. Sukhatme. Downwash-aware trajectory planning for large quadrotor teams. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[14] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.*, 219(C):40–66, 2015.

[15] M. Čáp, J. Gregoire, and E. Frazzoli. 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)*, 2016.

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

[17] Glenn Wagner and Howie Choset. M*: A complete multirobot path planning algorithm with optimality bounds. In Dejan Milutinović and Jacob Rosen, editors, *Redundancy in Robot Manipulators and Multi-Robot Systems*, pages 167–181. Springer, Berlin, Heidelberg, 2013.

[18] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *Proc. of the 19th National Conference on Innovative Applications of Artificial Intelligence*, IAAI'07, pages 1752–1759, 2007.