Integrated Motion Planning and Coordination for Industrial Vehicles

Marcello Cirillo and **Federico Pecora** and **Henrik Andreasson**
Center for Applied Autonomous Sensor Systems, Örebro University, Sweden
<name>.<surname>@oru.se

Tansel Uras and **Sven Koenig**
Department of Computer Science, University of Southern California, USA
{turas,skoenig}@usc.edu

Abstract

A growing interest in the industrial sector for autonomous ground vehicles has prompted significant investment in fleet management systems. Such systems need to accommodate on-line externally imposed temporal and spatial requirements, and to adhere to them even in the presence of contingencies. Moreover, a fleet management system should ensure correctness, i.e., refuse to commit to requirements that cannot be satisfied. We present an approach to obtain sets of alternative execution patterns (called trajectory envelopes) which provide these guarantees. The approach relies on a constraint-based representation shared among multiple solvers, each of which progressively refines trajectory envelopes following a least commitment principle.

Introduction

The industry standard approach to the management of fleets of autonomous vehicles still largely relies on fixed trajectories (Marshall, Barfoot, and Larsson, 2008) and manually specified ad-hoc traffic rules. This approach presents two drawbacks: First, traffic rules are not sufficient to guarantee deadlock-free coordination; Second, even small modifications to the environment require the definition of new trajectories and the manual update of the traffic rules, both of which are expensive and lengthy procedures.

An automated solution to autonomous fleet management should adhere to several requirements: (1) the individual motions of each vehicle should be feasible with respect to the vehicle's capabilities; (2) deadlocks and collisions must be avoided; (3) the system should be able to accommodate externally imposed temporal and spatial constraints.

We propose an integrated approach to fleet management which guarantees the achievement of kinematic, dynamic, temporal, and spatial requirements, as well as the absence of deadlocks and collisions. Our approach is designed to work online and has been experimentally validated both in simulation and on real vehicles. Finally, we prove the relevant formal properties of the system under reasonable conditions.

Representation

Our system is grounded upon the notion of trajectory envelopes (Pecora, Cirillo, and Dimitrov, 2012), that is, collections of spatial and temporal constraints on vehicle trajectories. Specifically, a trajectory envelope is composed of a *spatial envelope* and a *temporal envelope*. The former is a set $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ of spatial constraints on where a vehicle's reference point can be within a given map M of the environment. Each spatial constraint $\mathcal{S}_i$ is a set of linear inequalities defining a convex 2D polyhedron. Fig. 1 depicts an example of constraints on (x, y). To each $\mathcal{S}_i$ we associate a set of temporal constraints $\mathcal{T}_i$ of the form

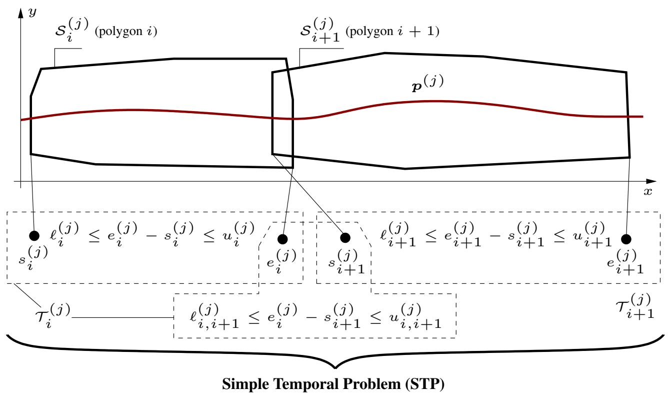

y
$\mathcal{S}_i^{(j)}$ (polygon i)
$\mathcal{S}_{i+1}^{(j)}$ (polygon $i+1$)
$\boldsymbol{p}^{(j)}$
x
$s_i^{(j)}$
$\ell_i^{(j)} \leq e_i^{(j)} - s_i^{(j)} \leq u_i^{(j)}$
$e_i^{(j)}$
$s_{i+1}^{(j)}$
$\ell_{i+1}^{(j)} \leq e_{i+1}^{(j)} - s_{i+1}^{(j)} \leq u_{i+1}^{(j)}$
$e_{i+1}^{(j)}$
$\mathcal{T}_i^{(j)}$
$\ell_{i,i+1}^{(j)} \leq e_i^{(j)} - s_{i+1}^{(j)} \leq u_{i,i+1}^{(j)}$
$\mathcal{T}_{i+1}^{(j)}$
Simple Temporal Problem (STP)

Figure 1: A trajectory envelope for vehicle j consisting of two sets of polyhedral and temporal constraints.

$$\ell_i \leq e_i - s_i \leq u_i \quad (1)$$

$$\ell_{i,i+1} \leq e_i - s_{i+1} \leq u_{i,i+1}, \quad (2)$$

where s_i (e_i) denotes the time in which the vehicle's pose begins (ceases) to be within the polyhedral constraint $\mathcal{S}_i$, and $\ell_i, \ell_{i,i+1}, u_i, u_{i,i+1} \in \mathbb{R}$ are fixed lower and upper bounds. Hence, (1) defines bounds on when the reference point of the vehicle is within the convex region specified by $\mathcal{S}_i$, while (2) defines bounds on when the reference point is within the spatial overlap between $\mathcal{S}_i$ and $\mathcal{S}_{i+1}$ (which is a convex set as well). The temporal envelope of a vehicle is the collection of temporal constraints $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$.

Definition 1. *A* trajectory envelope *is a pair $\mathcal{E} = (\mathcal{S}, \mathcal{T})$, where*

- *$\mathcal{S} = \bigcup_i \mathcal{S}_i$ is the* spatial envelope *of the vehicle, and*
- *$\mathcal{T} = \bigcup_i \mathcal{T}_i$ is the* temporal envelope *of the vehicle.*

A trajectory envelope is thus a set of spatial and temporal constraints on the position of a vehicle's reference point.

Let $\boldsymbol{p} : [0,1] \to \mathbb{R}^2 \times \mathbb{S}^1$ denote a path for a vehicle, in terms of positions and orientations of its reference point, parametrized using its arc lenght σ. Given a time history along the path $\sigma = \sigma(t)$, we refer to $\boldsymbol{p}(\sigma)$ as a trajectory. $\mathcal{E}$ contains the trajectory $\boldsymbol{p}(\sigma)$ if $\boldsymbol{p}(\sigma(t)) \in \mathcal{S}_i$ for all $t \in \mathcal{T}_i$. Given N vehicles, each with $\mathcal{E}^{(j)}, j \in \{1 \dots N\}$, the problem of finding $s_i^{(j)}$ and $e_i^{(j)}$ (for all i, j) that satisfy temporal constraints $\{\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(N)}\}$ is a Simple Temporal Problem (STP) (Dechter, Meiri, and Pearl, 1991) with variables

$$\boldsymbol{t} = \bigcup_{i,j} \{s_i^{(j)}, e_i^{(j)}\}.$$

A STP admits many solutions, each defining the amount of time $e_i^{(j)} - s_i^{(j)}$ during which vehicle j's reference point should be within the polyhedral constraints $\mathcal{S}_i^{(j)}$. A solution of this STP can be found in $\Theta(|\mathcal{S}|^3)$ with the Floyd-Warshall all-pairs-shortest-paths algorithm (Floyd, 1962), where $\mathcal{S} = \{\mathcal{S}^{(1)}, \dots, \mathcal{S}^{(N)}\}$.

Integrated Reasoning

The collection of trajectory envelopes $\{\mathcal{E}^{(1)}, \dots, \mathcal{E}^{(N)}\}$ for all vehicles constitutes a constraint network with temporal and spatial constraints (Fig. 2). This network is used as the common representation for reasoning by several modules:

- A *motion planner* adds an initial trajectory envelope $\mathcal{E}^{(j)} = (\mathcal{S}^{(j)}, \mathcal{T}^{(j)})$ for each vehicle j such that (1) the constraints in $\mathcal{S}^{(j)}$ cover one or more kinematically feasible paths that lead the vehicle from its current position to a goal position, and (2) the constraints in $\mathcal{T}^{(j)}$ bound the vehicle to traverse the polyhedra at least at the vehicle's minimum speed and at most at its maximum speed.
- A *coordinator* refines the envelopes $\{\mathcal{E}^{(1)}, \dots, \mathcal{E}^{(N)}\}$ of all vehicles so as to eliminate (1) trajectories which lead to collisions or deadlocks, as well as (2) those which do not satisfy the temporal constraints present in the network.
- A *vehicle executive* realizes the interface between vehicle controllers and the trajectory envelope representation: it selects an appropriate trajectory to be executed by each controller, and updates the trajectory envelopes with constraints representing the current progress of each vehicle. This propagates any mismatch between prescribed and executed trajectories on all vehicles in the fleet.
- A *trajectory smoother* further refines trajectory envelopes, by excluding those which are not feasible with respect to the dynamic constraints of each vehicle and the specific mechanics of the deployed controllers.
- A *controller* on board each vehicle synthesizes control actions according to a Model Predictive Control (MPC) scheme (Qin and Badgwell, 2003). The controller also feeds back the current status of a vehicle so that it can be accounted for by other modules.

Each algorithm alters the common constraint-based representation to prune out solutions that are infeasible from its particular point of view. All algorithms are used on-line, in a continuous control loop at three levels of abstraction: *coordination layer* (motion planner and coordinator), *vehicle execution layer* (vehicle executive and trajectory smoother), and *vehicle control layer*.

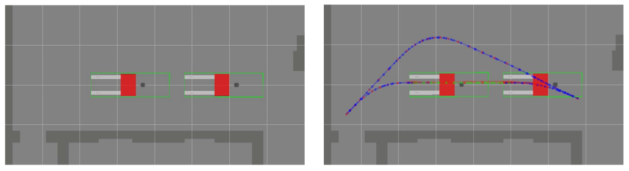

(a) (b)

Figure 3: Two vehicles need to switch places (a). A solution which considers the joint spatial constraints of the vehicles is required (b).

Motion Planning

Non-holonomic motion planning has been extensively studied in the past decades (LaValle, 2006). In particular, three families of methods are currently widely used: Probabilistic Roadmaps (PRMs) (Kavraki et al., 1996), Rapidly-exploring Random Trees (RRTs) (LaValle, 1998; Karaman and Frazzoli, 2011) and lattice-based motion planners (Pivtoraiko and Kelly, 2009). Our motion planner belongs to the latter family: lattice-based approaches combine the strengths of the other two with well studied classical AI graph-exploration algorithms. Here, the differential constraints are incorporated into the search space by means of pre-computed motion primitives which trap the motions onto a regular lattice. However, in our framework, a single-robot motion planner would not be enough. For instance, consider the problem represented in Fig. 3(a): if the two vehicles need to switch places without changing orientation, their individual motions would completely overlap and no temporal adjustment could avoid a collision. Therefore, we extended our planner to solve multi-robot problems. Multi-robot path planning is a very active research area, but existing solutions either depend on strongly simplifying assumptions (Wagner and Choset, 2011; Luna and Bekris, 2011), or cannot guarantee deadlock-free joint motions (ter Mors, 2011).

Single Vehicle. Given a model of vehicle maneuverabilty, the intuition behind lattice-based motion planning is to sample the state space in a regular fashion and to trap the motions of the vehicle on a lattice graph $G = \langle V, E \rangle$, that is, a graph embedded in a Euclidean space $\mathbb{R}^n$ which forms a regular tiling (Pivtoraiko, Knepper, and Kelly, 2009). Each vertex $v \in V$ represents a valid pose, or configuration, of the vehicle, while each edge $e \in E$ encodes a motion which respects the non-holonomic constraints of the vehicle. Our first aim is to generate time-independent motions, thus we focus only on kinematic constraints.

Definition 2. *A model* m *of a vehicle is a tuple* $\langle d, \Theta, \Phi, r, P, g \rangle$, *where* d *encodes the geometric dimensions of the vehicle,* Θ *a finite set of orientations,* Φ *a finite set of steering angles,* r *the resolution of an* $\mathbb{R}^2$ *regular grid,* P *a finite set of allowed motions which respect the kinematic constraints of the vehicle and* g *a cost function.*

Definition 3. *A valid* configuration *for a vehicle model* m= $\langle d_m, \Theta_m, \Phi_m, r_m, P_m, g_m \rangle$ *is a four dimensional state*

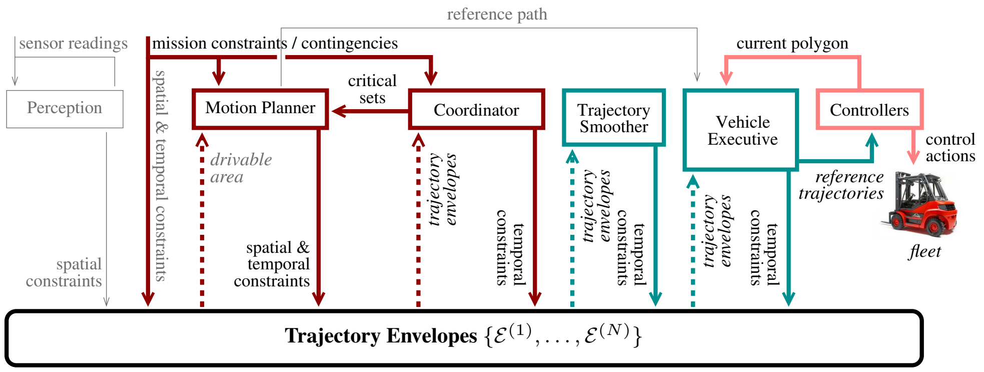


Figure 2: All algorithms cooperatively refine a common constraint network representing trajectory envelopes.

vector $c = \langle x, y, \theta, \phi \rangle$, where (x, y) lies on a grid of resolution r_m, $\theta \in \Theta_m$ and $\phi \in \Phi_m$.

Under the assumption of even terrain, we can design P_m (the set of motion primitives of model m) to be position-invariant.[1] A cost function g_m is associated to each $p \in P_m$. In our implementation, $g_m(p)$ is calculated by multiplying the distance covered by p to a cost factor which penalizes backwards and turning motions. A planning problem for the j-th vehicle is fully specified by a four-tuple $\langle m^{(j)}, c_s^{(j)}, c_g^{(j)}, M \rangle$, where $m^{(j)}$ is a vehicle model, $c_s^{(j)}$ is the start configuration of the vehicle, $c_g^{(j)}$ its goal configuration and M is a grid map of the environment with the same resolution of $r_{m^{(j)}}$ in which all known obstacles are represented. A *valid solution* to the planning problem is a collision-free sequence of motion primitives $\pi^{(j)} = (p_0^{(j)}, \ldots, p_n^{(j)})$ connecting $c_s^{(j)}$ to $c_g^{(j)}$. An optimal solution $\pi_{opt}^{(j)}$ is a valid solution with minimum cost.

Starting from $c_s^{(j)}$, the state space can be explored using efficient graph search algorithms (in our case, A^* or ARA^* (Likhachev, Gordon, and Thrun, 2003)). To speed up lattice exploration, we use a combination of heuristic functions: Euclidean distance, a pre-calculated distance table (Knepper and Kelly, 2006) (both admissible) and, for cluttered environments, we run a wavefront algorithm from the cell containing c_g, whose results we use to avoid useless exploration of blocked areas. As this last heuristic is not necessarily admissible for non-holonomic vehicles, we use it only in cluttered environments. The final heuristic value equals the maximum of the single heuristics employed.

Extension to Multiple Robots. Our planner is an extension of lattice-based motion planning to multi-robot systems, where we can consider vehicles with different models and also multiple vehicles with the same model. In the implementation presented in this paper, we impose the constraint that the grid r is identical for all vehicles. Definition 3 still holds for individual vehicles, but the search space of the joint plan is no longer a lattice graph. Here, each state in the search space represents a joint configuration of all vehicles:

Definition 4. *Given a set of N vehicles, a* state *is an ordered set $C = \{c^{(1)}, \cdots, c^{(N)}\}$, where each $c^{(i)} \in C$ is a valid configuration of vehicle i and there is no spatial overlap between any two configurations.*

[1]This assumption can be relaxed if the vehicle's controller can absorb minor perturbations or by means of a post-processing step.

Definition 5. *A multi-robot planning problem for N vehicles is fully specified by a tuple $\langle \mathcal{M}, C_s, C_g, M \rangle$, where:*

- *$\mathcal{M} = \{m^{(1)}, \cdots, m^{(N)}\}$ is an ordered set of vehicle models, such that $m^{(j)}$ is the model of vehicle j;*
- *$C_s = \{c_s^{(1)}, \cdots, c_s^{(N)}\}$ is an ordered set of start configurations, such that $c_s^{(j)}$ refers to the j-th vehicle;*
- *$C_g = \{c_g^{(1)}, \cdots, c_g^{(N)}\}$ is an ordered set of goal configurations, such that $c_g^{(j)}$ refers to the j-th vehicle;*
- *M is a grid map of the environment where all known obstacles are specified.*

A *valid global solution* to a multi-robot planning problem is an ordered set of sequences of motion primitives $\pi_{glob} = (\pi^{(1)}, \cdots, \pi^{(N)})$ such that $\pi^{(i)}$ connects $c_s^{(i)}$ to $c_g^{(i)}$ for every $1 \leq i \leq N$ and there exist at least one valid schedule which avoids collisions among vehicles. An optimal solution to the problem is a valid global solution which minimizes the sum of the costs for all vehicles. Given a multi-robot motion planning problem with N vehicles, a state corresponds to a point in the joint configuration space of the form $C = \{c^{(1)}, \cdots, c^{(N)}\}$, where $c^{(i)} = \langle x^{(i)}, y^{(i)}, \theta^{(i)}, \phi^{(i)} \rangle$. We generate the successors of a state C by selecting in turn each vehicle i and by applying to its configuration $\bar{c}^{(i)}$ all the applicable motion primitives in $P_{m^{(i)}}$, while considering the other vehicles as obstacles. Therefore, in each successor, all the individual configurations of vehicles remain unchanged but one. Each vertex $\hat{v}$ corresponds to a unique state $\hat{C}$ in the joint configuration space. The cost associated to $\hat{v}$ is equal to the cost of the shortest path from C_s to $\hat{C}$. The optimal solution is therefore a minimum-cost path in the search space from C_s to C_g. In our implementation, we explore the search space using A^* and ARA^*. Note that the single vehicle case is a special case of the multi-robot problem. Here, we use the same heuristic functions described above, and the heuristic value of a state is calculated as the sum of the heuristic values of the individual configurations. As the search space becomes rapidly very large, we also designed and implemented two methodologies for speeding up the search, whose description is outside the scope of this paper. However, it is important to note that neither prune valid solutions.

From timeless motions to trajectory envelopes. For each vehicle, the output of the motion planner is published to the common constraint network in the form of spatial and temporal constraints. First, a set of convex polyhedra is calculated which covers the path in the current map of the environment. The spatial constraints contain the positions and orientations computed by the motion planner, as well as adjacent positions and orientations obtained by "sweeping" the footprint of the vehicle within given displacement and turning parameters. This procedure gives controllers the freedom to spatially deviate from the reference trajectory while remaining within the spatial constraints. The result is an initial spatial envelope $\mathcal{S}^{(j)} = \{\mathcal{S}_1^{(j)}, \ldots, \mathcal{S}_n^{(j)}\}$ for each vehicle j. An example for five vehicles is shown in Fig. 8. Second, an initial temporal envelope $\mathcal{T}^{(j)}$ is calculated for each vehicle containing the constraints (1) and (2). The input to the procedure is the state information as provided by the motion planner and boundary conditions on initial and final velocities, steering velocity, speed and acceleration. Based on the maximum allowed velocity and distance between two states, a minimum transition time Δt_{min} is computed. This is used to compute the lower bounds in the temporal envelope $\mathcal{T}$. The maximum transition time Δt_{max} determines the upper bounds, and is computed by plugging in minimum boundary conditions. We impose a minimum speed $v_{\text{min}} = \epsilon > 0$: this allows vehicles to move arbitrarily slow, a condition which is needed to ensure schedulability of joint motions.

Coordination

The set of constraints $\mathcal{T}^{(j)}$ resulting from the motion planner constitutes a STP which admits, by construction, at least one solution (assignment of $\boldsymbol{t}$). Note, however, that since vehicles share the same floor space, it is also necessary to impose that trajectory envelopes do not overlap in both time and space (which would imply the possibility of collisions).

Definition 6. *A* conflict *is a pair of polyhedra $(\mathcal{S}_k^{(i)}, \mathcal{S}_m^{(j)})$, in the trajectory envelopes of vehicle i and j respectively, that overlap both spatially and temporally, i.e.,*

$$\mathcal{S}_k^{(i)} \cap \mathcal{S}_m^{(j)} \neq \emptyset \wedge \tag{3}$$

$$\left[s_k^{(i)}, e_k^{(i)}\right] \cap \left[s_m^{(j)}, e_m^{(j)}\right] \neq \emptyset. \tag{4}$$

Therefore, the solution of the STP may not be conflict-free, because there is no constraint in $\mathcal{T}$ that enforces the absence of conflicts. The coordinator is responsible for determining a set of temporal constraints $\mathcal{T}_a$ that refines the temporal envelopes so as to be conflict- and deadlock-free.

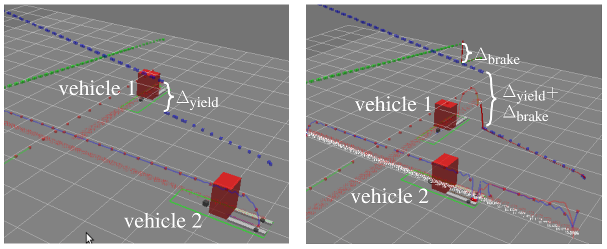
Definition 7. *Given the trajectory envelopes $\mathcal{E} = \{\mathcal{E}^{(1)}, \ldots, \mathcal{E}^{(N)}\}$ of N vehicles computed by the motion planner, a* coordination problem *consists of finding a set of constraints $\mathcal{T}_a$ that, if added to $\mathcal{E}$, eliminate all conflicts.*

Finding a set of additional constraints $\mathcal{T}_a$ that make $\mathcal{E}$ conflict-free can be cast as a Constraint Satisfaction Problem (CSP) (Tsang, 1993), where the variables are conflicts (see Definition 6). The values of these variables are temporal constraints that eliminate temporal overlap between polygons (*resolving constraints*). Finding a set of conflict-free trajectory envelopes requires exponential time, as it can be cast as a resource scheduling problem with maximum time-lags[2] (Brucker and Knust, 2006). It would indeed be also possible to employ de-centralized approaches to solve this problem. However, in order to guarantee global feasibility with respect to temporal and spatial constraints along with the absence of collisions and deadlocks, a distributed approach would also require exponential computation, either in the form of an exponential number of messages or of exponential message size (Faltings, 2006). Note also that while polynomial-time distributed algorithms can be used to guarantee safe navigation (Bekris et al., 2012), these cannot enforce adherence to temporal constraints like deadlines.

```
Function ScheduleTrajectories($\mathcal{E}$) : success or failure
1  static $\mathcal{T}_a = \emptyset$
2  $\mathcal{C} \leftarrow$ pairs of conflicting polyhedra
3  while $\mathcal{C} \neq \emptyset$ do
4      $(\mathcal{S}_k^{(i)}, \mathcal{S}_m^{(j)}) \leftarrow$ Choose($\mathcal{C}$, $h_c$) // (see Definition 6)
5      $\mathcal{R}_c = \left\{ s_k^{(i)} \geq e_m^{(j)}, e_k^{(i)} \leq s_m^{(j)} \right\}$
6      while $\mathcal{R}_c \neq \emptyset$ do
7          $r \leftarrow$ Choose($\mathcal{R}_c$, $h_r$)
8          $\mathcal{R}_c \leftarrow \mathcal{R}_c \setminus r$ // remove constraint $r$ from $\mathcal{R}_c$
9          $\mathcal{T}_a \leftarrow \mathcal{T}_a \cup r$ // add constraint $r$ to STP
10         if STP is consistent then
11             if ScheduleTrajectories($\mathcal{E}$) = failure
               then
12                 $\mathcal{T}_a \leftarrow \mathcal{T}_a \setminus r$
13             else return success
14         else $\mathcal{T}_a \leftarrow \mathcal{T}_a \setminus r$
15     return failure
16 return success
```

Algorithm ScheduleTrajectories() solves the trajectory scheduling problem with a standard CSP backtracking search. It is inspired by the Earliest Start Time Approach precedence-constraint posting algorithm (Cesta, Oddi, and Smith, 2002) for resource scheduling. The algorithm starts by collecting all conflicts (line 2). The assessment of condition (4) (possible temporal overlap) is performed by comparing the Earliest Time solutions of the STP. As usual in CSP search, the conflicts are ordered according to a most-constrained-first variable ordering heuristic. In our case, h_c gives preference to pairs of polygons that are spatially closer to other conflicting pairs (as earlier failures allow to prune large parts of the search tree). Once a conflict is chosen (line 4), its possible resolving constraints are identified (line 5). These are the values of the CSP's variables, and each is a temporal constraint that would eliminate the temporal overlap of intersecting polygons. Since conflict sets are pairs of polyhedra, there are only two ways to resolve the temporal overlap, namely imposing that the end time of polyhedron $\mathcal{S}_m^{(j)}$ is constrained to occur before the start time of polyhedron $\mathcal{S}_k^{(i)}$, or vice-versa. Again as is common practice in CSP search, the value (resolving constraint) to

[2]Floor space can be seen as a shared resource which is concurrently used by the vehicles when they traverse a polygon.



(a) (b)

Figure 4: Example of coordination (a), and of protracted yielding due to delay of a vehicle with precedence (b).

attempt first is chosen (line 7) according to a least constraining value ordering heuristic. h_r leverages temporal slack to choose the ordering that least affects the temporal flexibility of the time points. The algorithm then attempts to post the chosen resolving constraint into the STP (line 9). If the STP is consistent, then the procedure goes on to identify and resolve another conflict through a recursive call (line 11). In case of failure (line 14), the chosen resolving constraint is retracted from the STP and another value is attempted.

An example of coordination is shown in Fig. 4(a): the trajectory envelopes of two vehicles are modified to eliminate temporal overlap from spatially overlapping polyhedra; in particular, vehicle 2 is scheduled to yield for a time Δ_{yield} to vehicle 1. The figure shows one possible temporal profile that is conflict free — note that the trajectory envelope is refined by the coordinator, therefore many alternative temporal profiles for the two vehicles' trajectories exist. As we show below, commitment to a particular profile does not occur until a reference trajectory is dispatched to the vehicle controllers; also, other possible temporal profiles may be selected subsequently as a result of on-line contingencies.

Vehicle Execution Layer

The execution layer of our architecture comprises two modules, namely a vehicle executive and a trajectory smoother. The former computes reference trajectories for the vehicle controllers. Specifically, the controller operates using a fixed period of 60 ms, and for each period it requires a reference state and velocities. The input to the computation of reference trajectories consists of a reference path (from the motion planner), and temporal envelopes that have been refined by the coordinator (from the common representation). Each $\mathcal{T}_i^{(j)}$ of vehicle j's temporal envelope provides bounds on when vehicle j should reach or leave a spatial envelope $\mathcal{S}_i^{(j)}$. A reference trajectory is extracted by computing a solution to the temporal problem (committing to a particular assignment of times to polyhedra), and combining the obtained times with the reference path. This is done by first applying a linear interpolation to assign a velocity profile to the reference path (as the coordinator uses the initial temporal envelopes $\mathcal{T}^{(j)}$ to assign times, the required velocity change between envelopes may require too high accelerations). To ensure that the trajectory will be dynamically feasible, we apply a trajectory smoothing method, which employs an MPC based scheme (similar to the one used by the controller) to perform a simulated run where the computed control actions form the smoothed reference trajectory. Although we cannot guarantee the dynamic feasibility of the solutions generated by the coordinator, we have in practice never generated a solution which could not be executed.

Continuous Reasoning

The algorithms described above are computationally very different, ranging from exponential (motion planning and coordination) to low-order polynomial complexity. For this reason, the frequency of operation of each algorithm varies widely: vehicle controllers continuously solve a Quadratic Programming problem every 60 ms (16.7 Hz), and the vehicle executive updates the network and forwards reference trajectories at 10 Hz. Coordination is performed at most once per second. Finally, motion planning is only called into play when new goals for vehicles are available, and trajectory smoothing is triggered when motions are computed.

Coordinator-Motion Planning Loop

Motion planner and coordinator realize a more tight integration when needed. Given N vehicles, motions are first computed independently, resulting in N trajectory envelopes $\{\mathcal{S}^{(1)} \cup \mathcal{T}^{(1)}, \ldots, \mathcal{S}^{(N)} \cup \mathcal{T}^{(N)}\}$. If the addition of constraints that eliminate conflicts were completely delegated to the coordinator, conflicts could only be resolved by removing temporal overlap. However, there may be cases in which the only way to enforce the absence of conflict also involves removing spatial overlap. This is the case, for instance, when two vehicles are required to switch places (Fig. 3): the motion planner would most likely generate completely overlapping spatial envelopes for the two vehicles, thus eliminating the possibility of temporal adjustment. In situations such as these, the `ScheduleTrajectories()` algorithm fails. The failure is signaled to the motion planner, along with a *critical set* $\{i_1, \ldots, i_m\}$ of vehicles that should be considered for multi-robot motion planning. The result is joint motions for all m vehicles which include non-overlapping polygons in which vehicles can yield for others. The new envelopes are then temporally refined by the coordinator to enforce that vehicles yield to each other.

The problem of identifying sets of vehicles the joint motion planning of which yields successful coordination is solved as follows: each conflict that is visited (line 4 in `ScheduleTrajectories()`) is added to a set $\mathcal{C}$; let the number of occurrences of vehicle i in $\mathcal{C}$ be $\text{occ}(i, \mathcal{C})$. If coordination fails, a vector of vehicle indices is created and ordered by increasing $\text{occ}(i, \mathcal{C})$; the first two elements of the vector are provided to the motion planner; if subsequent coordination fails, joint motion planning is attempted on the vehicles whose indices are in position two and three of the vector; after all successive pairs of indices in the vector have been exhausted, the size of the set of indices is increased to three, and so on. This is to increase the computational burden of joint motion planning only to the extent necessary.

Coordinator-Vehicle Executive Loop

The vehicle executive also updates the temporal envelopes of all vehicles to reflect the current state of vehicles as execution progresses. These updates allow the coordinator to continuously propagate any deviations from the temporal profile. This is necessary because although the constraints $\mathcal{T}_a$ computed by the coordinator ensure the absence of collisions and deadlocks, the delay of vehicle j may induce other vehicles yielding to j to protract yielding. For example, suppose vehicle 1 is scheduled to proceed through the intersection before vehicle 2 (Fig. 4(a)), which is scheduled to stay idle for Δ_{yield}; an obstacle blocks vehicle 1 before the intersection (Fig. 4(b)), rendering vehicle 1 idle for Δ_{brake}; the $\mathcal{T}_a$ computed by the coordinator are still contained in the vehicles' trajectory envelopes, thus forcing vehicle 2 to wait for $\Delta_{\text{brake}} + \Delta_{\text{yield}}$ in its current position until vehicle 1 has passed the intersection. Here, it may be convenient to re-invoke the coordinator to compute a set of resolving constraints $\mathcal{T}'_a$ to replace $\mathcal{T}_a$. These constraints take into account that time has elapsed since the last computation, which may have altered the heuristic value h_r of a resolving constraint. This would be the case in the example above if vehicle 2's deadline had in the mean time become significantly closer; the coordinator would then resolve the threat of collision by giving precedence to vehicle 2, while forcing vehicle 1 to procrastinate its mission if it recovers from the delay to accommodate the new precedence. In general, the coordinator is re-invoked whenever further delays could compromise a mission deadline, so as to ensure that enough temporal slack is available at trajectory envelope generation. In the very rare occasions when spatial envelopes need re-calculation, mission deadlines could become unachievable. However, in this highly unlikely circumstance, our system is still guaranteed to notify failure and stop execution (see Theorem 2).

Coordination and vehicle execution are interleaved, and therefore vehicles are subject to changing reference trajectories. As the coordinator computes temporal bounds without considering vehicle dynamics, the trajectory smoother is responsible for re-computing the input to the j-th controller whenever $\mathcal{T}^{(j)}$ changes as a result of coordination.

Finally, it is necessary to account for the computational overhead required by motion planning and coordination when modeling the current state of the fleet in the constraint network. If coordination occurs at time t_{now}, the resulting constraints will eliminate conflicts under the assumption that vehicles have not changed state between t_{now} and $t_{now}+\Delta t$, where Δt is the execution time of the coordinator. This may lead to incorrect behavior. We address this issue by (1) enforcing a timeout Δt to the solvers, and (2) allowing vehicle executives to feed back the predicted state at time $t_{now}+\Delta t$. If the solvers are not be able to generate valid trajectory envelopes in the given Δt, a stopping signal is sent to all vehicles to avoid unpredictable behaviour. This is followed by a second invocation of the solvers with a longer Δt. Note that in our experiments this mechanism has never been triggered.

Formal Properties

We wish to provide an overall fleet management system that can guarantee to compute trajectories that are kinematically- and dynamically-feasible, conflict- and deadlock-free. We subject these requirements to three assumptions (which account for the continuous nature of space and time): (1) the map used by the motion planner is appropriately discretized given the operational conditions of the system; (2) the set of motion primitives for each vehicle is chosen appropriately; and (3) parallel motions are never *required* to solve a multi-robot motion planning problem.[3] Under these assumptions:

Lemma 1. *Given a multi-robot problem where parallel motions are not required, the motion planner is complete.*

Proof. Under the assumption that the discretization of the working space and the set of motion primitives of each vehicle allow for a solution, completeness follows from the fact that the algorithm performs a systematic search. □

Lemma 2. *If the motion planner finds a joint plan from C_s to C_g, the plan is guaranteed to be schedulable.*

Proof. Each edge in the search graph represents the application of a single motion primitive of a single vehicle, while the others are treated as obstacles. Thus, if v_{min} is arbitrarily small, a joint plan extracted from the search graph connecting C_s to C_g corresponds already to a valid schedule. □

Lemma 3. *The* `ScheduleTrajectories()` *algorithm is correct.*

Proof. By construction, the search algorithm resolves all collisions by sequencing overlapping polyhedra of different vehicles though temporal precedence constraints, hence guaranteeing the absence of collisions. Constraints (1) and (2) imply that no feasible trajectory can take infinite time, thus eliminating the possibility of a temporal profile which leads to deadlock. □

Lemma 4. *The* `ScheduleTrajectories()` *algorithm is complete.*

Proof. Follows from the fact that the algorithm performs a systematic search in the space of possible sequencing constraints for conflicting polyhedra. □

Theorem 1 (Correctness). *If motion planning and coorindation succeed, the resulting trajectory envelopes contain at least one trajectory that is kinematically feasible, conflict- and deadlock-free.*

Proof. We impose a timeout Δt on both motion planner and coordinator, and these solvers assume that the initial state of the vehicles is the one predicted for $t_{now} + \Delta t$. Also, if motion planning and/or coordination fail, a stopping signal is sent to all vehicles so that they abort execution of possibly conflicting trajectories. Hence, provided that the prediction is accurate, correctness follows from Lemmas 2 and 3. □

Theorem 2 (Completeness). *If a set of kinematically-feasible, conflict- and deadlock-free trajectories exists, the motion planner and coordinator will yield a non-empty set of trajectory envelopes.*

Proof. Follows from Lemmas 1, 2, 4, and from the fact that coordination failure will eventually lead to the exploration of joint-motion solutions among all vehicles if necessary. □

[3]Such instances correspond to situations in which no vehicle can move without another moving concurrently, and are exceptional — e.g., bumper-to-bumper traffic in a one-lane roundabout.

	1 vehicle [msec]		2 vehicles [msec]	
	SW1	*SW3*	*SW1*	*SW3*
OS	(2) 4 (7)	-	(128) 379 (2163)	
Cross	-	-	(59) 67 (82)	(61) 67 (74)

Table 1: Aggregated results for the motion planner test runs, expressed as (25th percentile) median (75th percentile).

Experimental Evaluation

We evaluate our system along two lines. First, we briefly validate the performance of the computationally intensive solving components, namely the motion planner and the coordinator. We then illustrate our system on industrial autonomous forklifts, and perform a qualitative comparison with state of the art approaches for fleet management.

Motion Planning

We present a short evaluation of our motion planner in two simulated setups, *Open Space* and *Crossing*. We employed two vehicle models, *SW1* and *SW3*, representing the same car-like vehicle. Both models have a grid resolution of 0.2 meters and $|\Theta| = 8$. In *SW1*, $|\Phi_{SW1}| = 1$ and $|P_{SW1}| = 224$, while in *SW3* $|\Phi_{SW3}| = 3$ and $|P_{SW3}| = 736$. A larger set of motion primitives entails a larger branching factor in the search space. When testing the full system, we never generated joint plans for more than two vehicles at a time. As we are interested in evaluating the planner in realistic situations, here we consider the same number of vehicles.

The *Open Space* scenario is a 20 by 20 meters obstacle-free space, where we used the *SW1* model. We generated two sets of 50 test runs. In each set we deployed, respectively, one and two vehicles in the environment, with randomly chosen start and goal configurations, and we solved them optimally using A^*. The aggregated execution times are summarized in Table 1 (upper row), where we report their median and the 25th and 75th percentile. Most of the single-vehicle instances were solved in less than 7 ms, while when two vehicles were present at the same time, the median time rose to 379 ms and 75% of the problems took less than 2 seconds to be solved. There were, however, 3 outliers which took more than 30 seconds to be completed.

In the *Crossing* scenario, two vehicles maneuver in close quarters to pass a crossing. The simulated environment has a size of 10 by 10 meters and the vehicles' start and goal configurations are placed at one of the ends of the corridors. We prepared 14 test runs, covering all possible start/goal combinations for two vehicles without symmetries. Fig. 5(a) shows the start configurations of one of our test runs, where the two vehicles are required to swap places: the joint solution found by the planner ensures that one of the vehicles moves aside to let the second one pass (Fig. 5(b)). We repeated the same test runs using models *SW1* and *SW3*, and we explored the search space using A^*. The results, in terms of execution times, are presented in Table 1. The calculation of the joint plans required in all runs less than a second, even when we used model *SW3* which presents a $|P_{SW3}| = 736$.

Coordination

The evaluation of the coordinator was performed with a benchmark of 900 problems. On an 50 by 50 meters obstacle free map, we pre-defined 80 poses, where the (x, y) coordinates of each pose correspond to one of 10 points distributed on a circle, 40 meters in diameter, and where the orientation θ is one of 8 pre-determined angles. The problems were divided into 9 sets, each corresponding to an increasing number of vehicles concurrently deployed in the map, from 2 to 10. For each set, we performed 100 runs. In each run, we randomly chose initial and final poses for the number of vehicles required, only avoiding that any two vehicles had the same starting or final (x, y) positions. To make the problems more difficult to solve, we also added temporal constraints imposing that the temporal distance between all initial polyhedra is zero, thus forcing all vehicles to start moving at the same time. This, combined with a non-zero minimum speed for all vehicles, is what allows some benchmark problems to be unsatisfiable (UNSAT). We measured the time required by the coordinator to find a conflict-free solution for each run, or to identify the problem as unsolvable. The results are shown in Fig. 6 (the percentage of satisfiable problems is shown on top of each set). As expected, solving time grows exponentially with the number of vehicles involved.

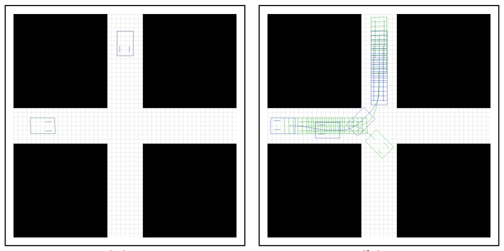

(a) (b)

Figure 5: Crossing scenario: 5(a) starting configurations in one of the test runs; 5(b) schedulable joint plan for the two vehicles.

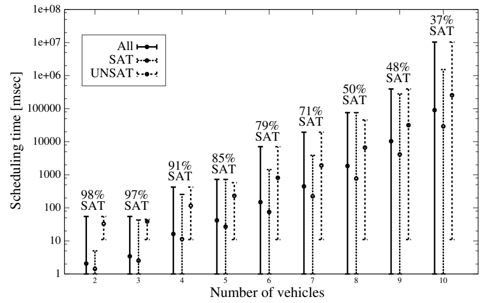


Figure 6: Quantitative evaluation of the coordinator.

Two features of these results are interesting. First, note that problem difficulty here is somewhat artificially inflated, as all vehicles are constrained to operate in a relatively small area at roughly the same time and they are also constrained to start moving at the same time. Even under these unlikely circumstances, the average resolution time remains under one second up to problems with 8 vehicles deployed. Second, the calculation of each set of discrete trajectory envelopes never takes more than 50 milliseconds (the most challenging problem here contains 94 polyhedra).

Test Cases with Industrial Forklifts

Our system has been deployed on two forklift platforms. The platforms have car-like kinematics and are controlled by velocity and steering commands. Three scenarios illustrate

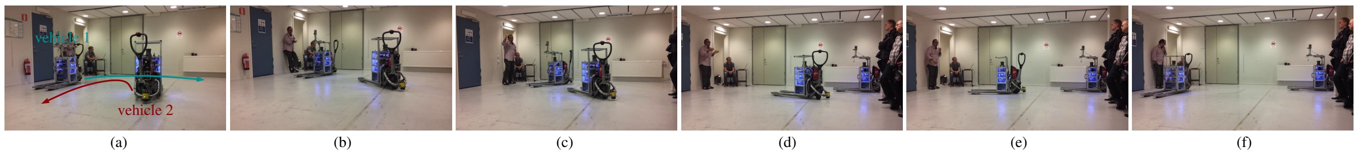


Figure 7: Second scenario: (a) vehicles 1 and 2 and their targets; (b) vehicle 1 is braked, causing vehicle 2 to yield; (c) the brake is released and vehicle 1 resumes motion; (d) vehicle 2 resumes motion; (e) vehicle 1 and (f) vehicle 2 reach their final poses.

the principal features of our approach. The first scenario involved repeatedly posting goal poses for the two forklifts in the test environment. The limited space leads to significant spatio-temporal overlap, thus incurring frequent yielding behavior. The second scenario is a variant of the first in which unforeseen contingencies were created by sending the vehicles brake commands (Fig. 7). We can contrast the scenario with state of the art approaches to coordination of large robot teams, e.g., the work of Kleiner, Sun, and Meyer-Delius (2011) in which robots are assumed to move on a grid. Although the method provides a means to resolve conflicts on-line, it cannot guarantee adherence to other temporal constraints, as the resolving strategies used are local. Also, grid motions would render impossible to satisfy the non-holonomic constraints of common industrial vehicles.

As a third scenario (see video attachment), we illustrate the ability of our system to recover from a coordination failure. The vehicles are instructed to switch poses, causing the motion planner to synthesize completely overlapping spatial envelopes. Consequently, the coordinator fails to find conflict-free envelopes and the multi-robot motion planner generates a new trajectory which ensure spatially disjoint polyhedra. Calculating local motions for each robot independently (Kleiner, Sun, and Meyer-Delius, 2011) would not safeguard against unschedulable situations. Also, methods which assign pre-defined priority levels to different robots (ter Mors, 2011), or use merit-based tokens to decide which agent should take initiative (Desaraju and How, 2011) cannot ensure deadlock-free situations.

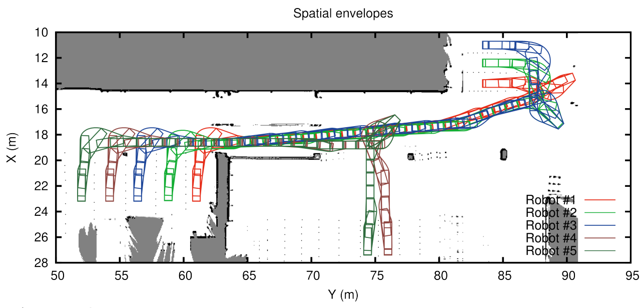


Figure 8: Spatial envelopes generated for a set of vehicles in a logistics environment.

We also performed a set of runs in a simulated industrial production site (using a physical simulation in Gazebo), which involve one to five vehicles in an area that affords long motions. The layout used in the simulation is part of a factory in which automatically guided vehicles transit along pre-defined paths. Conversely, in our simulation, the motions of the vehicles were computed on-line. Fig. 8 shows the occupancy map of the environment and the spatial envelopes for an initial set of target poses. The task was to traverse between production and storage areas to simulate the transportation of goods. To evaluate the performance of the system, we computed the amount of idle time for each vehicle. The analysis shows that idle time never exceeds 30%, even with 5 vehicles and under the assumption that motion planning and coordination are triggered only when previous goals are reached. This extremely unsophisticated form of task dispatching represents a worst case, as it forces vehicles to enter an idle state while new envelopes are computed, and is far less advanced than current industrial practice.

Conclusions and Future Work

We have presented a fleet management system which adheres to several key requirements distilled from industrial applications, among which completeness and correctness. Our approach is grounded on a common constraint-based representation of trajectories, called trajectory envelopes. Trajectory envelopes represent spatial and temporal constraints on coordinated trajectories for all vehicles in the fleet. Several solvers, each working on a different aspect of the overall problem, continuously refine the trajectory envelopes to remove infeasible solutions. The overall system, as well as its component solvers, have been tested both in simulation and with autonomous industrial forklifts. Our work opens several interesting avenues for future research. A first, interesting possibility would be the integration of new solvers in our system, which can impose additional constraints, e.g., high-level task allocation. Another important aspect that we intend to explore is how to realize a tighter integration between motion planner and coordinator, as we believe that an intertwined approach could improve the quality of the solutions. Also, from the motion planner perspective, it would be very interesting to investigate the application of state of the art multi-robot path planning algorithms, both optimal (Wagner and Choset, 2011) and suboptimal (Röger and Helmert, 2012), to the lattice state space – this requires adapting these algorithms to make them take into account the kinematic constraints of the vehicles. Finally, we also intend to study strategies to further reduce idle time and increase operations efficiency.

Acknowledgments. The research at Örebro University was supported by project "Safe Autonomous Navigation" (SAUNA), funded by the Swedish Knowledge Foundation (KKS). The research at USC was supported by NSF under grant number IIS-1319966 and ONR under grant number N00014-09-1-1031. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

References

Bekris, K. E.; Grady, D. K.; Moll, M.; and Kavraki, L. E. 2012. Safe distributed motion coordination for second-order systems with different planning cycles. *Int. Journal of Robotics Research (IJRR)* 31(2).

Brucker, P., and Knust, S. 2006. *Complex Scheduling*. Springer.

Cesta, A.; Oddi, A.; and Smith, S. F. 2002. A constraint-based method for project scheduling with time windows. *Journal of Heuristics* 8(1):109–136.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Desaraju, V., and How, J. 2011. Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Faltings, B. 2006. Distributed constraint programming. In Rossi, F.; van Beek, P.; and Walsh, T., eds., *Handbook of Constraint Programming*. Elsevier. 699–729.

Floyd, R. W. 1962. Algorithm 97: Shortest path. *Communication of the ACM* 5:345–348.

Karaman, S., and Frazzoli, E. 2011. Sampling-based algorithms for optimal motion planning. *The Int. Journal of Robotics Research* 30(7):846–894.

Kavraki, L.; Svestka, P.; Latombe, J.; and Overmars, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation* 12(4):566–580.

Kleiner, A.; Sun, D.; and Meyer-Delius, D. 2011. Armo: Adaptive road map optimization for large robot teams. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

Knepper, R. A., and Kelly, A. 2006. High performance state lattice planning using heuristic look-up tables. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press.

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems* 16.

Luna, R., and Bekris, K. E. 2011. Push and swap: fast cooperative path-finding with completeness guarantees. In *Proc. of the 22nd Int. Joint Conf. on AI (IJCAI)*.

Marshall, J.; Barfoot, T.; and Larsson, J. 2008. Autonomous underground tramming for center-articulated vehicles. *Journal of Field Robotics* 25(6-7):400–421.

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 Int. Conf. on Intelligent Robots and Systems (IROS)*.

Pivtoraiko, M., and Kelly, A. 2009. Fast and feasible deliberative motion planner for dynamic environments. In *Proc. of the ICRA Workshop on Safe Navigation in Open and Dynamic Environments: Application to Autonomous Vehicles*.

Pivtoraiko, M.; Knepper, R. A.; and Kelly, A. 2009. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics* 26(3):308–333.

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

Röger, G., and Helmert, M. 2012. Non-optimal multi-agent pathfinding is solved (since 1984). In *Proc. of the Fifth Annual Symp. on Combinatorial Search (SoCS)*.

ter Mors, A. 2011. Conflict-free route planning in dynamic environments. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego.

Wagner, G., and Choset, H. 2011. M*: A complete multi-robot path planning algorithm with performance bounds. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.