A Constraint-Based Approach for Multiple Non-Holonomic Vehicle Coordination in Industrial Scenarios

Federico Pecora and **Marcello Cirillo**
Center for Applied Autonomous Sensor Systems
Örebro University, SE-70182 Sweden
`<name>.<surname>@oru.se`

Abstract

Autonomous vehicles are already widely used in industrial logistic settings. However, applications still lack flexibility, and many steps of the deployment process are hand-crafted by specialists. Here, we preset a new, modular paradigm which can fully solve logistic problems for AGVs, from high-level task planning to vehicle control. In particular, we focus on a new method for multi-robot coordination which does not rely on pre-defined traffic rules and in which feasible and collision-free trajectories are calculated for every vehicle according to mission specifications. Also, our solutions can be adapted on-line to exogenous events, control failures, or changes in mission requirements.

Introduction

Industrial actors involved in the development of autonomous vehicles (e.g., autonomous forklifts for warehouses) are constantly interested in decision support tools which could improve the flexibility and the performance of their products. Atlas-Copco[1] (Larsson, Appelgren, and Marshall, 2010), Kiva Systems[2], INRO[3] (Thomson and Graham, 2011) and Kollmorgen[4], among others, aim to achieve complete automation in Autonomous Ground Vehicle (AGV) deployments. Although it is current practice to employ automated solutions in several aspects of logistics automation, many key parts of the deployment phase are still ad-hoc and manual. For instance, the definition of AGV paths is often done off-line, and these paths are hand crafted for each different setting. Also, large-scale industrial deployments of AGVs rarely include more than very crude heuristics to optimize mission scheduling. Another limitation of current industrial solutions is the resolution of spatial conflicts, which is often performed off-line through manually synthesized traffic rules, whose correctness cannot be formally proved. Other fallacies of real systems include the lack of support for resources and an often only partial support for on-line mission constraint posting (e.g., changed deadlines, new requirements, collapses of resource availability).

When automated solving components are used in industrial AGV deployments, these are usually not integrated. For instance, it is often the case that path planning is de-coupled from trajectory generation, or that the allocation of vehicles to destinations does not depend on the trajectory that will actually be followed by the vehicles. This leads to inefficiencies in the quality of the solutions and reduced flexibility in dealing with contingencies. Furthermore, the methodology for assessing how many AGVs are necessary for a particular deployment typically consists of what-if analyses on simulated scenarios. This analysis becomes more cumbersome and, especially, less accurate if many de-coupled solving modules are employed.

In this paper, we introduce a system which strives to facilitate all phases of deployment of AGVs in real settings. Our approach is modular, in that it can be applied "partially" or in "pieces", depending on the requirements of the particular deployment at hand. For instance, AGV paths may be automatically generated by a path planner (as is the case in this paper), or the routes could be manually decided by a field specialist (as is often the case in industrial settings). The same principle applies to task planning, which can be automated or manually decided by human operators (in this paper, the specific task planning algorithm is omitted). To achieve this, the modules rely on a shared, constraint-based representation of the overall problem, and each module refines this representation from "its own" point of view.

Related Work

Many of the problems underlying the automation of task and motion planning for industrial vehicles have been addressed in research. As a result, important advancements have been achieved in addressing separate parts of the overall problem. Algorithms such as M^* (Wagner and Choset, 2011), an extension of the classical A^* to multi-robot systems, and the work of Luna and Bekris (2011), whose focus is a new method for multi-robot path planning which is computationally efficient and complete, are recent examples of promising theoretical results. A new system for the coordination of large multi-robot teams has been presented by Kleiner, Sun, and Meyer-Delius (2011). The authors propose a system that generates an overall, optimal road map configuration. However, in this work the agents are assumed as moving on a grid, and the local motions are calculated for each robot independently from the motions of other robots.

A common approach for multi-robot path planning which

[1]`http://www.atlascopco.com`
[2]`http://www.kivasystems.com/`
[3]`http://www.inro.co.nz/`
[4]`http://www.kollmorgen.com`

usually guarantees fast results is the assignment of priority levels to different robots. This can be seen as an improved version of hand-coded traffic rules, but cannot ensure deadlock-free situations. An example of an algorithm which relies on this paradigm is presented by ter Mors (2011). The overall system can find optimal, conflict-free routes in low polynomial time, but relies on a pre-defined roadmap shared by all agents for path planning. Desaraju and How (2011) further extend the idea of prioritized path planning, by substituting the pre-defined priority levels with a merit based token, which is passed among agents. Once a robot has planned its own path, it circulates it to the other team members, which in turn update their trajectories.

In recent years, a number of approaches to multi-robot coordination have been presented which rely on pre-defined paths. Examples include the work of Kleiner, Sun, and Meyer-Delius (2011), whose algorithm is resolution complete and can be easily applied to situations in which a large number of agents is moving. However, the overall coordinated motions lack flexibility, as time is considered only implicitly in configurations along the paths. Therefore, the final result cannot take into account motion delays, or explicit temporal constraints imposed on the single agents and their positions over time.

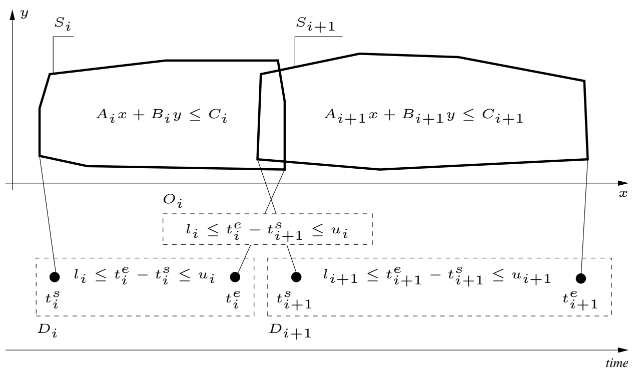
A Constraint-Based Approach

A trajectory is a sequence of points and an associated temporal profile, which specifies exactly when the vehicle will be in a certain point. Instead of reasoning in terms of one trajectory, in our approach we reason in terms of *temporal* and *spatial constraints* on trajectories. The collection of spatial and temporal constraints on one vehicle's trajectory is called a *trajectory envelope*. We can describe the overall mission planning problem (which will be defined precisely shortly) as a Constraint Satisfaction Problem (Tsang, 1993, CSP) where variables represent vehicles, their values represent possible trajectories that they should execute, and constraints are spatial and temporal requirements on these trajectories. A solution to the overall problem is therefore an assignment of trajectories to vehicles such that none of the requirements on trajectories is violated.

Trajectory envelopes are the key representational elements used to express and solve our problem. More precisely, (see also figure 1):

Definition 1. *A* trajectory envelope *is a triple $\langle \mathcal{S}, \mathcal{D}, \mathcal{O} \rangle$ where*

- $\mathcal{S} = \{S_1, \ldots S_n\}$ *is a set of linear* spatial constraints *in the form $A_i x + B_i y \leq C_i$; each set S_i of spatial constraints specifies a convex region in the map within which the vehicle must be contained;*
- $\mathcal{D} = \{D_1, \ldots D_n\}$ *is a set of linear* temporal constraints *in the form $l_i \leq t_i^e - t_i^s \leq u_i$, where t_i^s (t_i^e) represents the time at which the vehicle enters (exits) the area specified by the set of spatial constraints S_i; these constraints provide bounds on when the vehicle is within the convex region specified by S_i;*
- $\mathcal{O} = \{O_1, \ldots O_{n-1}\}$ *is a set of linear* temporal constraints *in the form $l_i \leq t_i^e - t_{i+1}^s \leq u_i$; these constraints provide bounds on when the vehicle is within the (convex) spatial overlap between S_i and S_{i+1}.*



y
S_i
S_{i+1}
$A_i x + B_i y \leq C_i$
$A_{i+1} x + B_{i+1} y \leq C_{i+1}$
x
O_i
$l_i \leq t_i^e - t_{i+1}^s \leq u_i$
$l_i \leq t_i^e - t_i^s \leq u_i$
t_i^s
t_i^e
$l_{i+1} \leq t_{i+1}^e - t_{i+1}^s \leq u_{i+1}$
t_{i+1}^s
t_{i+1}^e
D_i
D_{i+1}
time

Figure 1: A trajectory envelope consisting of two spatio-temporal polygons.

Problem Statement

Given N vehicles, we define the overall mission planning problem in our scenario as follows:

Definition 2. *A* mission planning problem *is a tuple $\langle M, \mathcal{I}, \mathcal{G}, \mathcal{T}, \mathcal{V} \rangle$, where*

- *M is a metric map of the environment;*
- *$\mathcal{I} = \{l_1, \ldots l_N\}$ is a set of coordinates in the map specifying the* initial location *of all vehicles;*
- *$\mathcal{G} = \{G_1, \ldots G_m\}$ is a set of* goals *in the form $\langle k, s, g \rangle$, each specifying that k loads must be transported from location s to location g;*
- *$\mathcal{T} = \{T_1, \ldots T_n\}$ is a set of* temporal constraints *on $\mathcal{G} \cup \mathcal{I}$; these constraints are in the form $l_i \leq t_x - t_y \leq u_i$, where t_x and t_y are start/end timepoints of a goal or initial position;*
- *$\mathcal{V} = \{V_1, \ldots V_N\}$ are the capacities of the vehicles (maximum amount $\in \mathbb{N}$ of load each vehicle can carry).*

Finding a solution to a mission planning problem is decomposable into three parts; first, compute an allocation of vehicles to goals which achieves the necessary displacement of loads to places; second, compute the trajectory envelopes $\mathcal{E}$ for each vehicle; third, synthesize a set of temporal constraints $\mathcal{T}_{\text{col}}$ imposing that spatio-temporal polygons intersect either only in space, or only in time, or not at all. Note that a solution to a mission planning problem in fact represents sets of possible trajectories for each vehicle.

Our approach is based on four functional modules, namely *task planning*, *trajectory planning*, *trajectory scheduling*, and *control*. All modules output constraints capturing some aspect of the mission's requirements. Together, these constraints define trajectory envelopes for all vehicles, and provide a global representation of the mission, from its high-level goals to the specific trajectories vehicles must execute to achieve these goals. Each of the four modules progressively refines the representation, imposing increasingly specific requirements:

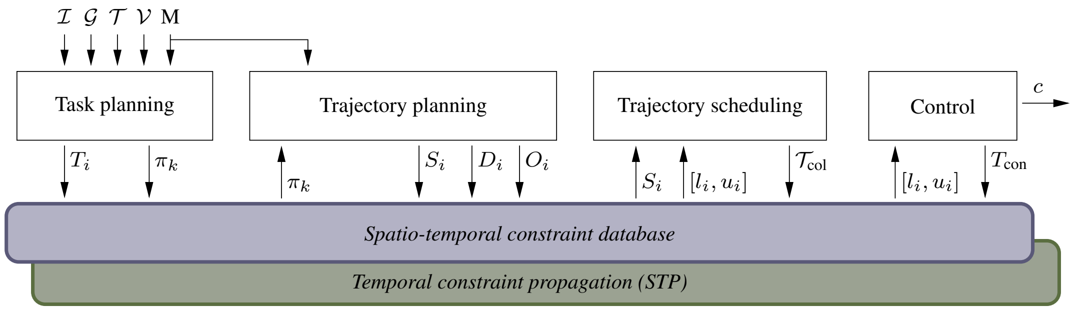

$\mathcal{I}$ $\mathcal{G}$ $\mathcal{T}$ $\mathcal{V}$ M
Task planning
Trajectory planning
Trajectory scheduling
Control
c
T_i
π_k
π_k
S_i
D_i
O_i
S_i
$[l_i, u_i]$
$\mathcal{T}_{\text{col}}$
$[l_i, u_i]$
T_{con}
Spatio-temporal constraint database
Temporal constraint propagation (STP)

Figure 2: Overall information flow of the four solving modules.

1. **Task planning** decides goal locations for currently available vehicles, therefore constraining the start and end points of vehicle trajectories. Also, task planning may impose quantified temporal requirements on these trajectories, e.g., "vehicle A must reach goal (x, y) before time t", or "vehicle A must reach its goal before vehicle B".
2. **Trajectory planning** decides locations that should be visited in-between goals for each vehicle. Crucially, these are not simple paths between the initial and goal positions, rather they are trajectory envelopes, i.e., temporally-constrained sets of spatial constraints.
3. **Trajectory scheduling** imposes further temporal constraints which ensure that the trajectory envelopes of different vehicles do not intersect in time and space. As explained below, this is a hybrid form of spatio-temporal reasoning based on state-of-the-art scheduling techniques.
4. The **Control** module employs a kinematic and geometric model of a vehicle to generate and follow one specific trajectory that lies within the trajectory envelope obtained as a result of the above modules. This results in appropriate control signals for the vehicle *or* in the selection of a new trajectory within the trajectory envelope in the case that the currently selected trajectory does not adhere to the constraints.

Note that commitment to a specific trajectory in the above scheme is performed only at the control level, and that the controller is given the constraints within which it can select one of many trajectories to follow. As we will show, the way in which these constraints are posted and propagated by the first three modules ensures that for each vehicle there exists a trajectory, within the constraints, which is feasible with respect to *all vehicles*. Also, due to the particular type of hybrid temporal-spatial scheduling performed by the trajectory scheduling module, the selection of mutually-compatible trajectories by all vehicles can be done in polynomial time.

All communication between the four modules occurs through a *spatio-temporal constraint database* (see figure 2), which contains variables and constraints defining an overall CSP. The variables of this problem are spatio-temporal polygons, and the constraints are spatial or temporal relations defining the trajectory envelopes. Note that the temporal constraints in $\mathcal{D}$ and $\mathcal{O}$ constitute a Simple Temporal Problem (Dechter, Meiri, and Pearl, 1991, STP), which is solvable in cubic time through the Floyd-Warshall (Floyd, 1962) temporal constraint propagation algorithm. The algorithm computes the lower and upper bounds $[l_i, u_i]$ of all timepoints given the constraints in the database, and is triggered every time a constraint is added to the database. Thus, through temporal constraint propagation, the bounds of all timepoints t_i of the spatio-temporal polygons are maintained at all times consistent with the temporal constraints that are present in the spatio-temporal database. Note also that if a temporal constraint is added which invalidates previously existing temporal constraints, the constraint database can detect this though a propagation failure, and thus rejects this constraint. This feature of temporal constraint propagation is employed in the trajectory scheduling to search for temporal constraints that avoid collisions between vehicles.

For convenience, we will refer to the sets $\mathcal{S}$ and $\mathcal{D} \cup \mathcal{O}$ as the *spatial* and *temporal envelopes* of a trajectory, respectively. Also, we refer to the i-th set of spatial and temporal constraints $\{S_i \cup D_i \cup O_i\}$ on a trajectory envelope as a *spatio-temporal polygon*. Two spatio-temporal polygons i and j are *spatially overlapping* if $S_i \cap S_j \neq \emptyset$. Temporal overlap is less straightforward: since the underlying STP maintains bounds on the timepoints of spatio-temporal polygons, temporal overlap must be assessed by choosing an earliest start time for the timepoints. Specifically, two spatio-temporal polygons i and j are said to be *temporally overlapping in the earliest start time solution* if $[l_i^s, l_i^e] \cap [l_j^s, l_j^e] \neq \emptyset$. Note that two spatially and temporally overlapping polygons belonging to trajectory envelopes of different vehicles entail that the vehicles may collide.

Solving a Mission Planning Problem

It is often the case in real-world deployments that a particular task allocation strategy, i.e., a task planning module, is given and cannot be substituted. This is due to the often very domain-specific objective functions, preferences and characteristics of the application scenario (e.g., a milk packaging factory vs. an underground mine). For this reason, in the following sections we omit details about task planning and focus on modules (2–4). Consequently, we assume for the purposes of the following description that a task planner has decided, for each goal $G = \langle s, g, k \rangle$, a high-level plan that achieves the displacement of k loads from s to g by an appropriate set of vehicles. Each element of this plan is in the form $\pi_k = \langle i, f, s, g \rangle$, indicating that vehicle i should load an amount $f \leq V_i$ of load in s and transport it to g. Obviously, f can also be equal to 0, when s represents the initial position of a vehicle and g its first load pick-up location.

We can now define the solution to the mission planning problem as follows:

Definition 3. *A* solution *to a mission planning problem $\langle M, \mathcal{I}, \mathcal{G}, \mathcal{C}, \mathcal{V} \rangle$ with N vehicles is a triple $\langle \Pi, \mathcal{E}, \mathcal{C} \rangle$ where*

- $\Pi = \{\pi_1, \ldots, \pi_p\}$ *is a set of high-level plans which achieve the goals in $\mathcal{G}$;*
- $\mathcal{E} = \{\langle \mathcal{S}_1, \mathcal{D}_1, \mathcal{O}_1 \rangle \ldots \langle \mathcal{S}_N, \mathcal{D}_N, \mathcal{O}_N \rangle\}$ *is a set of trajectory envelopes where*
  - $\mathcal{S}_i$ *is a set of spatial envelopes for the trajectory of vehicle i;*
  - *for every $\langle i, f, s, g \rangle$ in the high-level plan Π, $\mathcal{S}_i$ contains a sequence of spatial polygons $\langle S_1, \ldots, S_n \rangle$ where S_1 contains location s, S_n contains location g, and each S_j spatially overlaps S_{j+1};*
  - $\mathcal{D}_i$ *and $\mathcal{O}_i$ are sets of temporal constraints defining the temporal envelope of the trajectory for vehicle i; these constraints impose that overlapping spatial polygons also overlap in time;*
- $\mathcal{C} = \mathcal{T} \cup \mathcal{T}_{col}$ *is a set of temporal constraints between the start/end timepoints of any pair of spatio-temporal polygons in $\mathcal{E}$; this set contains the constraints $\mathcal{T}$ expressing the initial temporal requirements of the mission planning problem, as well as a set of constraints $\mathcal{T}_{col}$ which ensures that the intersection of spatio-temporal polygons for different vehicles is either only spatial, or only temporal, or neither (i.e., these constraints disallow collisions).*

Trajectory Planning

In order to obtain trajectory envelopes, we first employ a lattice-based planner to generate kinematically feasible paths for the (non-holonomic) vehicles in the mission planning problem. A lattice can be seen as a generalization of a grid: instead of using perpendicular lines, the state-space is discretized by repeating the same primitive set of connecting edges. We start from a set of kinematically acceptable motion primitives which can be repeated over and over to obtain a directed graph. Obviously, the graph need not be completely specified from the start, and can be progressively built during search. The graph is then efficiently explored using deterministic, theoretically sound algorithms. In our case, we chose to rely on the classic A^* (Hart, Nilsson, and Raphael, 1968) for optimal path generation[5], and on one of its most efficient anytime versions, ARA^* (Likhachev, Gordon, and Thrun, 2003), which can provide provable bounds on sub-optimality.

Our approach is inspired by existing lattice-based path planners, as the ones successfully used in real world application by Pivtoraiko, Knepper, and Kelly (2009) and by Urmson, Anhalt, and others (2008).

[5]The resulting path is optimal wrt the choice of the set of primitive motions and to the granularity with which the lattice is built.

Each vertex of the lattice represents a pose of the vehicle in the form $\langle x, y, \theta \rangle$, where x and y are coordinates on a grid of a pre-determined resolution, and $\theta \in \Theta$ is the vehicle orientation, where Θ is the set of pre-selected possible orientations for a specific vehicle model. For instance, in the experimental runs presented in this paper, the grid resolution is always equal to 0.2 meters, Θ is a set of 16 angles, equally spaced between π and $-\pi$, and each vertex is connected to 15 others through pre-calculated, kinematically feasible motion primitives. In our setup, the cost is based on the distance covered by each edge of the lattice, multiplied by a cost factor that penalizes backwards and turning motions.

Using off-line computation, it is possible to speed up the exploration of the lattice in environments with obstacles in two ways. First, as each edge is the instantiation of a pre-calculated motion template, we can pre-compute for each primitive the cells which the vehicle will partially or totally occupy during the motion. This way, obstacle detection can be efficiently performed on-line, by checking the occupancy level of each cell in the grid-partitioned environment. Second, a more informed heuristic function can be pre-computed and stored in a lookup table (Knepper and Kelly, 2006) by saving the minimum cost to connect two poses in a specific range (10 meters, in the experimental runs presented in this paper). This proves to be a much more efficient heuristic than simple Euclidean distance, as it uses the kinematic model of the non-holonomic vehicle to factor in maneuvering costs. Both functions are however admissible, and they entail optimal solutions when used with A^*.

When the environment presents obstacles, a third heuristic function is also used. Regardless of the pose of the vehicle, each cell in the environment is associated with a value represented by the distance from the goal in a 8-connected graph. All three heuristic functions are evaluated when a new vertex of the lattice is expanded, and we always use the higher in value. Clearly, the resulting heuristic function is not admissible in environments with obstacles. This, however, is not a big drawback in practical applications, where our goal is to obtain drivable and kinematically feasible, albeit sub-optimal, paths. Also, in real settings (as the one described below), we preferably employ ARA^* to explore the lattice, in order to speed up the computation, therefore relinquishing optimality anyway.

Recall that a trajectory envelope is defined as a set of temporally constrained spatio-temporal polygons. The starting point for computing the spatial constraints $\mathcal{S}_i$ for vehicle i's trajectory envelope is a path obtained through the path planning strategy outlined above. Then, the computed spatial envelope is used together with the path and the minimum and maximum speeds of the particular vehicle to determine the temporal envelope of the trajectory (i.e., the temporal constraints $\mathcal{D}_i$ and $\mathcal{O}_i$). These two procedures are described in the following paragraphs.

Spatial envelope generation. For each vehicle, waypoints are sampled along the path obtained by the path planning algorithm. The sampling procedure is incremental, and works as follows:

1. select the first two points of the path;
2. build a convex polygon around the vector defined by these two points whose shape is the bounding box of the vehicle, centered in the first point;
3. grow the sides of the polygon outwards, stopping the growth of each side when it intersects with an obstacle or when a threshold on growth has been reached;
4. select two points along the path immediately outside the polygon, and go to step (2).

The resulting sequence of polygons is such that (1) the path is completely covered by polygons, and (2) each polygon intersects the next one[6]. The resulting polygons are used to define the spatial envelope $\mathcal{S}_i$ for each vehicle i. All spatial envelopes are then added to the spatio-temporal constraint database.

Temporal envelope generation. Again starting from the first point along a vehicle's path, the path is traversed to compute the distance covered by the vehicle while traveling in each spatial polygon $S_j \in \mathcal{S}_i$. These distances are used to compute the temporal bounds within which the vehicle can possibly occupy each polygon and each area of polygon intersection. For this computation, we employ two constant speeds $(v_{\text{min}}, v_{\text{max}})$ corresponding to the minimum and maximum desired speeds for the vehicle. For each spatial polygon S_j we thus obtain a pair of bounds $[l_j, u_j]$ restricting the temporal distance between its start and end timepoints (t^s_j, t^e_j), as well as a pair of bounds $[l'_j, u'_j]$ restricting the distance between the end time t^e_j of spatial polygon S_j and the start t^s_{j+1} of spatial polygon S_{j+1}. Together, all these constraints constitute the spatial envelope $\mathcal{D}_i \cup \mathcal{O}_i$ of the trajectory of the i-th vehicle, and are added to the spatio-temporal constraint database.

Trajectory Scheduling

The spatio-temporal polygons generated by the trajectory planning module impose vehicles to be in certain (convex) regions within certain temporal bounds. In order to complete the synthesis of the solution to the mission planning problem, further constraints must be added (the set $\mathcal{T}_{\text{col}}$) in order to prune out of the solution trajectory envelopes in $\mathcal{E}$ those trajectories that lead to collisions. This problem is cast as a CSP whose variables are sets of spatio-temporal polygons which have a non-empty spatial and temporal intersection. The values of these variables are temporal constraints which separate these temporally concurrent, spatially overlapping polygons in time. In other words, the trajectory scheduling module resolves concurrent use of floor space by altering *when* different vehicles occupy spatially overlapping polygons. This results in temporal constraints that disallow the concurrent occupation of overlapping polygons by more than one vehicle at a time.

The reduction of the trajectory scheduling problem to a CSP is inspired by the ESTA precedence-constraint posting algorithm (Cesta, Oddi, and Smith, 2002) for resource

[6]Polygon intersection is not guaranteed with this procedure, which, however, gives very good results in practice.

```
Function SolveESTA($\mathcal{E}$) : success or failure
1  static $\mathcal{T}_{\text{col}} \leftarrow \emptyset$
2  repeat
3      conflicts $\leftarrow \{\langle (t^s_i, t^e_i), (t^s_j, t^e_j)\rangle \in \mathcal{E}$ :
4                  $[l^s_i, l^e_i] \cap [l^s_j, l^e_j] \neq \emptyset \wedge S_i \cap S_j \neq \emptyset\}$
5      if $conflicts \neq \emptyset$ then
6          MCS $\leftarrow$ Choose($conflicts$, $h_{var}$)
7          resolvers $\leftarrow \{(t^e_i, t^s_j) : D_i, D_{j\neq i} \in$ MCS$\}$
8          while $resolvers \neq \emptyset$ do
9              $(t^e_i, t^s_j) \leftarrow$ Choose($resolvers$, $h_{val}$)
10             resolvers $\leftarrow$ resolvers $\setminus (t^e_i, t^s_j)$
11             STP $\leftarrow$ STP $\cup (0 \leq t^s_j - t^e_i \leq \infty)$
12             if STP is consistent then
13                 $\mathcal{T}_{\text{col}} \leftarrow \mathcal{T}_{\text{col}} \cup (0 \leq t^s_j - t^e_i \leq \infty)$
14                 if SolveESTA($\mathcal{E}$) = failure then
15                     $\mathcal{T}_{\text{col}} \leftarrow \mathcal{T}_{\text{col}} \setminus (0 \leq t^s_j - t^e_i \leq \infty)$
16                 else return success
17 until $conflicts = \emptyset$
```

scheduling. The algorithm is a CSP-style backtracking search (see algorithm `SolveESTA()`). It starts by collecting all pairs of spatio-temporal polygons that overlap both spatially and temporally (line 3–4). These conflicts are the variables of the CSP, and as usual in CSP search, ordered according to a most-constrained-first variable ordering heuristic (h_{var}) — the rationale being that it is better to fail sooner rather than later so as to prune large parts of the search tree. Once a conflict is chosen, its possible resolvers are identified (line 7). These are values of the CSP's variables, and each is a temporal constraint to be imposed between the pair of spatio-temporal polygons that would eliminate their temporal overlap. Note that since conflicts are pairs of spatio-temporal polygons, there are only two ways to resolve the temporal overlap, namely imposing that the end time of one spatio-temporal polygon is constrained to occur before the start time of the other, or vice-versa. Again as is common practice in constraint-based reasoning, the resolver to attempt first is chosen (line 9) according to a least constraining value ordering heuristic (h_{val}) — the rational being that the value which leaves most options open for future choices should be given precedence. The algorithm then attempts to post the chosen resolving constraint into the spatio-temporal constraint database (line 11). If the underlying STP is still consistent, then the procedure goes on to identify and resolve another conflict through a recursive call (line 14). In case of failure (line 15), the chosen value is retracted from the spatio-temporal constraint database and another value is attempted.

Clearly, the efficiency of the search for resolving constraints depends on how well-informed the value and variable ordering heuristics are. In our specific case, we employ two heuristics which take into account both the temporal and the spatial features of the trajectory scheduling problem. The heuristic h_{var} employed for variable ordering gives preference to the pairs of spatio-temporal polygons that are spatially closer to other conflicting pairs. The idea of this heuristic is that conflicts that are "close" to other conflicts are more likely to be the most difficult to solve, as the possible choices for resolving these conflicts will depend on how other conflicts are resolved.

As a value ordering heuristic, we follow the method used by Cesta, Oddi, and Smith (2002), whereby the temporal bounds $\left[l_i^{s/e}, u_i^{s/e}\right]$ and $\left[l_j^{s/e}, u_j^{s/e}\right]$ of the start/end timepoints of the chosen pair of spatio-temporal polygons are analyzed to determine which ordering least restricts the temporal slack of the intervals.

From Envelopes to Vehicle Control

The trajectory scheduling module performs the last step in defining trajectory envelopes that solve the mission planning problem. Every vehicle's control module must at this point select one particular trajectory (i.e., a path and a speed profile) within the vehicle's trajectory envelope to execute. However, it is important to note that the particular trajectory chosen by each vehicle's controller depends on which trajectory other vehicles have chosen, as trajectories of different vehicles are temporally dependent.

The presence of temporal dependencies between trajectories entails that vehicle controllers must communicate their choice to other controllers. This choice can be seen as a set of temporal constraints T_{con}, which is added to the shared constraint database. Here, we leverage an important feature of the STP underlying our constraint database: in a fully propagated and consistent STP, i.e., one in which the bounds $[l_i, u_i]$ of all timepoints t_i have been updated to reflect the constraints, there exist two specific assignments of times to timepoints that are temporally consistent, namely the *earliest time assignment (ET)* and the *latest time assignment (LT)*. The former is obtained by choosing the lower bound l_i for all timepoints, and the latter by choosing the upper bound u_i for all timepoints. Therefore, we can immediately obtain the fastest and slowest speed profiles for all vehicles.

Our vehicle control scheme consists in a model predictive controller (Qin and Badgwell, 2003) which synthesizes control outputs to the vehicle. These outputs enable the vehicle to follow the given trajectory, both with respect to the spatial constraints $\mathcal{S}_i$ and with respect to a particular solution to the temporal constraints in $\{\mathcal{D}_i \cup \mathcal{O}_i\}$. Having selected a particular speed profile, this means that a controller must enter and exit spatial polygons exactly at the times prescribed in the particular speed profile selected for that vehicle (e.g., the fastest speed profile). Whenever these times cannot be achieved, vehicle controllers must revise their trajectories and compute new control outputs. Fortunately, to compute an allocation of times which is different from the ET or LT, it is sufficient to impose one constraint which models the desired allocation of one timepoint, and this can be achieved in polynomial time. As a result of propagation, the new ET allocation will clearly be temporally feasible. Indeed, even more interestingly, numerous alternative, globally consistent speed profiles can be computed before hand, each of which reflects one specific time in which vehicles should enter and exit each spatial polygon on their trajectory.

Experimental Evaluation

We now present an experimental validation of the two central modules of our approach, namely trajectory planning and trajectory scheduling. We validate the modules both qualitatively and quantitatively, with a special focus, in the quantitative analysis, on the performance of the trajectory scheduling algorithm. All test runs were performed in simulation. The kinematic model employed in all the experiments is that of a Linde H50D forklift.

Qualitative Evaluation

A single run in an industrial scenario was performed to qualitatively assess the feasibility of the approach in a realistic setting. For this purpose, we used a real map of an underground mine (courtesy of Atlas-Copco Drilling Machines, see figure 3), where we deployed 7 identical vehicles with pre-assigned tasks. Each task consisted in an initial and final pose for one of the vehicles, in the form $\{\langle x_i, y_i, \theta_i\rangle, \langle x_f, y_f, \theta_f\rangle\}$.

The overall run consisted of three phases. First, our lattice-based path planner generated in parallel individual kinematically feasible paths. Second, the paths were sampled to calculate the spatial envelopes for each vehicle. Assuming that all the forklifts would start moving at the same time, and defining the minimum and maximum desired speed in the tunnels (v_{min},v_{max}), we thus obtained a temporal envelope for each vehicle. The `SolveESTA()` algorithm was then invoked to generate a solution to the mission planning problem. As explained above, the algorithm identified all the conflicts in space and time over the temporally and spatially constrained polygons, and solved them by imposing additional temporal constraints $\mathcal{T}_{col}$.

In this specific run, considering the initial temporal and spacial envelopes for each single vehicle, the scheduler identified three groups of conflicting polygons (shaded in figure 3). Each conflict reflects the fact that, with only the temporal constraints stemming from the desired v_{min} and v_{max} of each vehicle along its nominal path, two or more vehicles would be "allowed" to be in overlapping areas at the same time, if they chose some particular velocity profiles.

The scheduler's solution consisted of 13 temporal constraints. This resulted in revised bounds for each of the spatio-temporal polygons such that in any consistent execution (e.g., the earliest start time, or fastest, execution) vehicles yield to each other appropriately in order to avoid collisions.

Extracting a specific trajectory for execution other than the earliest and latest time trajectories takes about 250 milliseconds. The total time required to generate the scheduled trajectory envelopes was less than 40 seconds: the paths were generated using the ARA^* algorithm, with a cut-off time of 5 seconds, and then used to grow a total of 140 polygons for the 7 vehicles, while trajectory scheduling took less than 34 seconds.

Quantitative Evaluation

To evaluate our approach in a more thorough and quantitative way, we generated a benchmark set of 900 trajectory scheduling problems. On an obstacle free map of

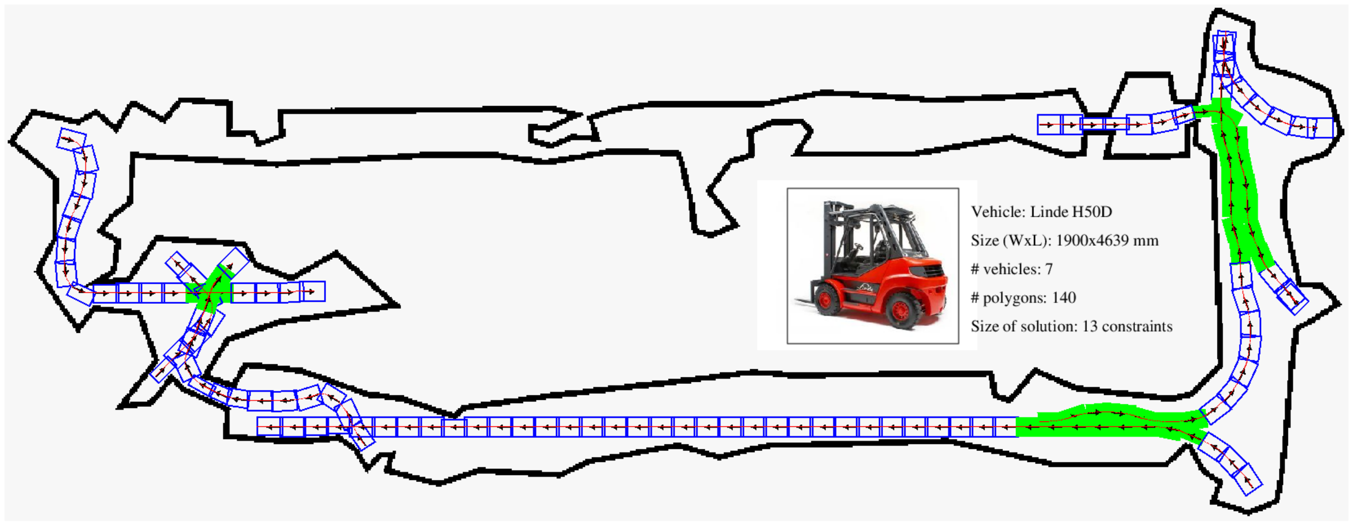


Figure 3: A solution to a mission planning problem involving seven vehicles in an underground mine. Spatio-temporal polygons involved in critical sets during trajectory scheduling are shaded.

width and length of 50 meters, we pre-defined 80 poses $\{\langle x_1, y_1, \theta_1\rangle, \ldots, \langle x_{80}, y_{80}, \theta_{80}\rangle\}$, where the (x, y) coordinates of each pose correspond to one of 10 points spatially distributed on a circle, 40 meters in diameter, and where the orientation θ is one of 8 pre-determined angles. Each pose could be chosen as initial of final pose for a vehicle, with the only constraint that the (x, y) coordinates of the two poses should be different.

The experimental evaluation was performed by defining 9 test sets, each corresponding to an increasing number of vehicles concurrently deployed in the environment, from 2 (the minimum number of vehicles whose spatial and temporal envelopes could generate conflicts) to a maximum of 10. For each set, we performed 100 test runs, as follows. In each run, we randomly chose initial and final poses for the number of vehicles required, only avoiding that two or more vehicles had the same starting or final (x, y) positions. Once generated, the paths were used to obtain the spatial and temporal envelopes with $(v_{\text{min}}, v_{\text{max}}) = (0.05, 15)$ meters per second. In order to make the problems difficult to solve for the scheduler, we also added temporal constraints imposing that the temporal distance between all initial spatio-temporal polygons 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.

Again, we focused on analyzing the trajectory scheduling efficiency, so we measured the time required by the scheduler to find a conflict-free solution for each run, or to identify the problem as unsolvable. The results are shown in figure 4. As expected, scheduling time grows exponentially with the number of vehicles involved.

Two features of these results are interesting. First, note that problem difficulty in this benchmark is somewhat artificially inflated as all vehicles are constrained to operate in an area which is 40-meter diameter circle at roughly the same

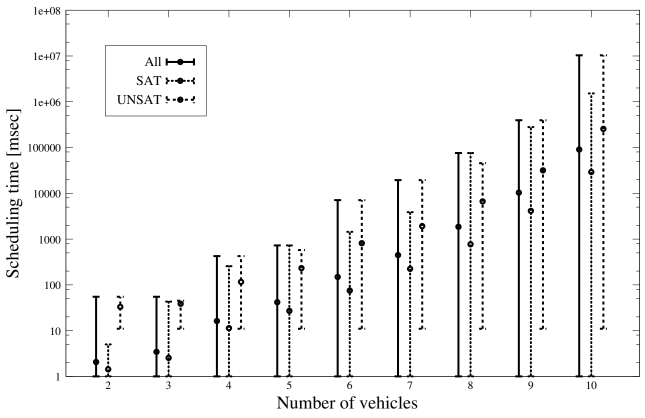


Figure 4: Quantitative evaluation of the trajectory scheduler.

time. Moreover, all vehicles are constrained to start moving at the same time, as all starting spatio-temporal polygons are constrained to occur at the same time and a minimum velocity of zero is not possible. Even under these rather unlikely circumstances, the average resolution time remains under one second up to problems in which we deployed 8 vehicles. Second, recall that the solutions obtained through the scheduling procedure represent a trajectory envelope for each vehicle. Two single trajectories, the earliest time and latest time trajectory, can be extracted for all vehicles in linear time (in the number of polygons). This is, even for the most difficult problem, an operation which takes less than 10 milliseconds. Furthermore, even if one vehicle controller decides it must stray from its current chosen trajectory, the calculation of another trajectory for all other vehicles is also a matter of milliseconds, as it can be done in cubic time. Specifically the most challenging problem of our benchmark contains 94 polygons, and we can extract a new trajectory for all vehicles in less than 50 milliseconds.

Comparing heuristics. In a comparison against a random choice variable and/or value ordering heuristic, the proposed h_{var} and h_{val} lead to dramatically better performance. We have also compared h_{var} to a heuristic commonly used in scheduling which employs only temporal features of the constraint network to determine the most constrained variable (Cesta, Oddi, and Smith, 2002). Our spatio-temporal heuristic lead to better performance of the backtracking search algorithm, although a complete comparison is necessary to establish whether the effect is due to the particular problem structure of the benchmark.

Conclusions and Future Work

This paper presents a new approach to multiple vehicle coordination in industrial environments. The framework is composed of four different modules for solving logistics problem for AGVs. The modules progressively refine a constraint-based representation of the overall problem, taking into account high-level task planning goals and temporal requirements to ultimately obtain commands for vehicle control. Our approach is engineered in a way that single modules can be used independently, thus providing the flexibility required in industrial settings.

We have focused on the two central modules of our framework, namely trajectory planning and trajectory scheduling. Our main contribution lies in multi-robot system coordination: instead of relying on ad-hoc traffic rules, or predefined priority levels, we used an on-line scheduler to synchronize the movements of the AGVs. Our scheduler allows maximum flexibility, as vehicle trajectories can be globally adapted to exogenous events, control failures, or changed mission requirements. The two modules are evaluated both qualitatively and quantitatively, proving that our approach can be used on-line, and that the results can be immediately employed by low-level controllers.

Our future work will focus on the full development of the remaining two modules, the task planner and a robust model predictive controller, for one or more specific industrial settings. Also, we will explore the use of different heuristics for trajectory scheduling. Finally, we intend to test the full framework and/or parts of it in real industrial scenarios to demonstrate the benefit of our new paradigm in terms of deployability and efficiency.

Acknowledgments. This work is supported by project “Safe Autonomous Navigation” (SAUNA), funded by the Swedish Knowledge Foundation (KKS). The Authors wish to thank Dimiter Driankov, Dimitar Dimitrov and Simone Fratini for their support and useful comments.

References

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)*.

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

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

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)*.

Larsson, J.; Appelgren, J.; and Marshall, J. 2010. Next generation system for unmanned lhd operation in underground mines. In *Proc. of the Annual Meeting and Exhibition of the Society for Mining, Metallurgy & Exploration (SME)*.

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. 2011. Efficient and complete centralized multi-robot path planning. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

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.

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)*.

Thomson, J., and Graham, A. 2011. Efficient scheduling for multiple automated non-holonomic vehicles using a coordinated path planner. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*.

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

Urmson, C.; Anhalt, J.; et al. 2008. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics* 25(8):425–466.

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)*.