On Mission-Dependent Coordination of Multiple Vehicles under Spatial and Temporal Constraints

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

***Abstract*— Coordinating multiple autonomous ground vehicles is paramount to many industrial applications. Vehicle trajectories must take into account temporal and spatial requirements, $e.g.$, usage of floor space and deadlines on task execution. In this paper we present an approach to obtain sets of alternative execution patterns (called *trajectory envelopes*) which satisfy these requirements and are conflict-free. The approach consists of multiple constraint solvers which progressively refine trajectory envelopes according to mission requirements. The approach leverages the notion of least commitment to obtain easily revisable trajectories for execution.**

I. Introduction

Coordination of multiple Autonomous Ground Vehicles (AGVs) in industrial applications is still largely performed off-line through manually synthesized traffic rules and/or local trajectory adjustments during execution. This entails several drawbacks, among which the lack of provably correct solutions and limited flexibility with respect to changed mission requirements, delays, or vehicle failures.

In recent years, several approaches have been proposed to address these problems. Algorithms such as M^* [1], an extension of the classical A^* to multi-robot systems, and the work of Luna and Bekris [2], whose focus is a new computationally efficient and complete method for multi-robot path planning, are recent examples of promising theoretical results (albeit limited to path planning). A system for the coordination of large multi-robot teams has been presented by Kleiner et al. [3]. 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.

Other computationally efficient approaches to multi-robot coordination leverage the assignment of pre-defined priority levels to different robots [4], or the use of a merit-based token, which is passed among agents to decide which should take initiative [5]. Both approaches can be seen as improved versions of hand-coded traffic rules, but cannot ensure deadlock-free situations.

In this work, we present an approach to multiple nonholonomic vehicle coordination, which is robust to delays and changed mission requirements. The approach consists of multiple constraint solvers which progressively refine trajectories according to spatial and temporal requirements. The approach leverages the notion of least commitment to obtain easily revisable trajectories for execution which are deadlock- and collision-free.

This work is partially funded by the Swedish Knowledge Foundation (KKS) under project "Safe Autonomous Navigation" (SAUNA).

II. A Constraint-Based Approach

In this paper, we deal with vehicles subject to nonholonomic constraints. In order to simplify the notation, we will adopt a model of a (rear-wheel drive) car-like vehicle, although the presented approach can be used in combination with other types of systems ($e.g.$, articulated vehicles). The kinematic model of a car-like vehicle is given by[1]

$$\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{v}) = (v\cos(\theta), v\sin(\theta), \frac{v}{l}\tan(\phi), \omega), \quad (1)$$

where (x, y) are the coordinates of the middle of the rear wheel-axis (which is our reference point for the vehicle) in the world frame, θ is the angle of the platform (in the world frame), ϕ is the steering angle, and l is the distance between the middle point of the front and rear wheel axes [6]. $\boldsymbol{q} = (x, y, \theta, \phi) \in \mathbb{R}^4$ and $\boldsymbol{v} = (v, \omega) \in \mathbb{R}^2$ denote the state and control vector, respectively. We will use $(\cdot)^{(j)}$ to indicate that variable $(\cdot)$ is associated to the j-th vehicle. When there is no ambiguity, the superscript (j) will be omitted.

Let $\boldsymbol{p}^{(j)} : [0, 1] \rightarrow \mathbb{R}^2$ denote a *path* for the reference point of vehicle j, parametrized using its arc length σ. Hence, $\boldsymbol{p}^{(j)}(0)$ denotes the starting position, and $\boldsymbol{p}^{(j)}(1)$ denotes the final position of the reference point. Given a time history along the path $\sigma = \sigma(t)$, we refer to $\boldsymbol{p}^{(j)}(\sigma)$ as a trajectory.

Definition 1: A trajectory $\boldsymbol{p}^{(j)}(\sigma)$ is *feasible* if

- it can be obtained from the evolution of (1) for suitable initial conditions $x(0)$, $y(0)$, $\theta(0)$, $\phi(0)$ and bounded control inputs $\underline{v} \leq v \leq \overline{v}$ and $\underline{\omega} \leq \omega \leq \overline{\omega}$
- the steering angle is bounded ($\underline{\phi} \leq \phi \leq \overline{\phi}$)
- for every pose of the vehicle along the path, its geometric model does not intersect any known obstacle.

In other words, a feasible trajectory is such that perfect execution in nominal conditions can be achieved in the presence of bounds on the steering angle and obstacles in the environment. We use $\underline{(\cdot)}$ and $\overline{(\cdot)}$ to denote lower and upper bounds on $(\cdot)$.

[1]For conciseness of notation, we will use $\boldsymbol{x} = (x_1, \ldots, x_n)$ to denote the elements of a column vector $\boldsymbol{x}$.

A. From Trajectories to Trajectory Envelopes

Since vehicles share a common floor space, coordination is necessary to ensure the absence of collisions and deadlocks. Current practice in both industry and research is to assume that trajectories for all vehicles are computed (either on-line or off-line) and committed to *before* coordination occurs. This early commitment thus implies that collisions and deadlocks are dealt with locally, *e.g.*, delaying one vehicle to avoid a collision, or following pre-set traffic rules to avoid deadlocks. The locality of these adjustments to vehicle trajectories entails that overall requirements of the fleet of vehicles cannot be guaranteed. Our approach aims to overcome this difficulty by leveraging constraint-based techniques that calculate the necessary requirements on trajectories which avoid collisions and deadlocks. The overall strategy is to inform the controller of each vehicle about these requirements so that it can synthesize control actions that are guaranteed to result in collision- and deadlock-free execution. This is achieved through the use of *trajectory envelopes*, which are essentially collections of spatial and temporal constraints on $\boldsymbol{p}^{(j)}$ and $\sigma^{(j)}(t)$.

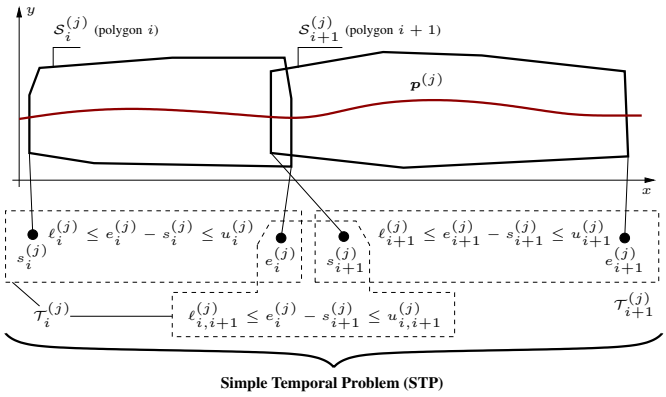

Fig. 1. A trajectory envelope for vehicle j consisting of two sets of polyhedral and temporal constraints (the θ and ϕ dimensions of the vehicle's state are omitted).

More specifically, a *spatial envelope* for vehicle j is a set of n sets of polyhedral constraints $\mathcal{S}^{(j)} = \{\mathcal{S}_1^{(j)}, \ldots, \mathcal{S}_n^{(j)}\}$ on the state variables of the j-th vehicle. The i-th set of linear constraints $\mathcal{S}_i^{(j)}$ is defined as the intersection of finitely many half-spaces. Fig. 1 depicts an example of constraints on (x, y). To each $\mathcal{S}_i^{(j)}$ we associate a set of temporal constraints $\mathcal{T}_i^{(j)}$ in the form

$$\ell_i^{(j)} \le e_i^{(j)} - s_i^{(j)} \le u_i^{(j)} \tag{2}$$

$$\ell_{i,i+1}^{(j)} \le e_i^{(j)} - s_{i+1}^{(j)} \le u_{i,i+1}^{(j)}, \tag{3}$$

where $s_i^{(j)}$ ($e_i^{(j)}$) denotes the time in which the vehicle's pose begins (ceases) to be within the polyhedral constraints $\mathcal{S}_i^{(j)}$, and $\ell_i^{(j)}, \ell_{i,i+1}^{(j)}, u_i^{(j)}, u_{i,i+1}^{(j)} \in \mathbb{R}$ are fixed lower and upper bounds. Hence, (2) defines bounds on when the reference point of the vehicle is within the convex region specified by $\mathcal{S}_i^{(j)}$, while (3) defines bounds on when the reference point is within the spatial overlap between $\mathcal{S}_i^{(j)}$ and $\mathcal{S}_{i+1}^{(j)}$ (which is a convex set as well).

We can, at this point, define a *trajectory envelope* for the j-th vehicle as $\mathcal{E}^{(j)} = \left(\mathcal{S}^{(j)}, \mathcal{T}^{(j)}\right)$, where

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

A trajectory envelope is thus a set of spatial and temporal constraints on the position of a vehicle's reference point. $\mathcal{E}^{(j)}$ contains $\boldsymbol{p}^{(j)}(\sigma)$ if $\boldsymbol{p}^{(j)}(\sigma(t)) \in \mathcal{S}_i^{(j)}$ for all $t \in \mathcal{T}_i^{(j)}$. The problem of finding $s_i^{(j)}$ and $e_i^{(j)}$ (for all i and j) that satisfy the temporal constraints $\mathcal{T} = \{\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(N)}\}$ is a Simple Temporal Problem (STP) [7] with variables

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

A STP admits many solutions, each of which defines the amount of time $e_i^{(j)} - s_i^{(j)}$ during which vehicle j's reference point should be within the polyhedral constraints $\mathcal{S}_i^{(j)}$. It is convenient to define a discretized version of these durations, which we refer to as the *discrete trajectory envelope* of vehicle j, that is, a sequence of polyhedral constraints

$$\langle \mathcal{S}_{I_1}^{(j)}, \mathcal{S}_{I_2}^{(j)}, \ldots, \mathcal{S}_{I_D}^{(j)} \rangle,$$

where I is a sequence of indexes of constraints. For example, suppose that a solution to the STP is (refer also to Fig. 1 assuming $i = 1$)

$$s_1^{(j)} = 7, \quad e_1^{(j)} = 13, \quad s_2^{(j)} = 10, \quad e_2^{(j)} = 15.$$

This solution implies that vehicle j's reference point will be in $\mathcal{S}_1^{(j)}$ for $e_1^{(j)} - s_1^{(j)} = 6$ time units, that it will be in $\mathcal{S}_2^{(j)}$ for $e_2^{(j)} - s_2^{(j)} = 5$ time units, and that for $e_1^{(j)} - s_2^{(j)} = 3$ time units it will be in both polyhedra. Assuming a discretization of these durations with $\Delta t = 1$, we obtain the sequence

$$I = \langle 1, 1, 1, \{1, 2\}, \{1, 2\}, \{1, 2\}, 2, 2 \rangle.$$

Hence, the reference point of vehicle j has to satisfy $\mathcal{S}_1^{(j)}$ at the first three discrete times, both $\mathcal{S}_1^{(j)}$ and $\mathcal{S}_2^{(j)}$ at the following three discrete times, and $\mathcal{S}_2^{(j)}$ at the last two discrete times.

Note that a solution of this STP can be found in $\Theta((2|\mathcal{S}|)^3)$ with the Floyd-Warshall all-pairs-shortest-paths algorithm [8], where $\mathcal{S} = \{\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(N)}\}$.

B. Problem Definition

The problem of finding a particular path $\boldsymbol{p}^{(j)}$ in the spatial envelope of vehicle j is a Constraint Satisfaction Problem (CSP) [9]. As we have seen, the problem of finding a time profile $\sigma^{(j)}(t)$ for this path is a STP whose constraints are the temporal envelope of vehicle j. Together, the solutions of these two CSPs constitute one trajectory for vehicle j. As mentioned, we wish to avoid committing to specific trajectories until a vehicle's controller has to synthesize control signals to actually displace the vehicle. The method we propose in this paper relies on the use of several solvers which can reason about the properties of trajectory envelopes

without having to commit to one particular trajectory. The first of these properties is feasibility:

Definition 2: A trajectory envelope $\mathcal{E}^{(j)} = \left(\mathcal{S}^{(j)}, \mathcal{T}^{(j)}\right)$ is *feasible* if it contains at least one feasible trajectory $\boldsymbol{p}^{(j)}(\sigma)$.

The feasibility of one trajectory envelope does not depend or alter the feasibility of other vehicles' trajectory envelopes. However, since vehicles share the same floor space, it is important that these trajectory envelopes do not overlap in both time and space (which would imply the possibility of two vehicles colliding).

Definition 3: A *conflict set* is a set of indexes $\mathcal{C}$ that contains all pairs $(i^{(j)}, k^{(m)})$ for which there is both spatial and temporal overlap, or formally

$$\mathcal{S}_i^{(j)} \cap \mathcal{S}_k^{(m)} \neq \emptyset \wedge \tag{4}$$

$$\left[s_i^{(j)}, e_i^{(j)}\right] \cap \left[s_k^{(m)}, e_k^{(m)}\right] \neq \emptyset. \tag{5}$$

Definition 4: A set of trajectory envelopes $\mathcal{E} = \bigcup_j \mathcal{E}^{(j)}$ is *feasible* if $\mathcal{C} = \emptyset$ for some solution $\boldsymbol{t}$ to the STP.

The problem of finding an overall set of trajectory envelopes that is feasible can be understood as an optimization problem whose decision variables contain the spatial, temporal and control variables (over all vehicles). Note that a solution to this problem is deadlock-free by definition[2] as constraints (2) and (3) imply that no feasible trajectory can take infinite time, thus eliminating the possibility of a temporal profile which leads to a deadlock.

Finding a set of feasible trajectory envelopes requires exponential time, as it requires resource scheduling with maximum time-lags[3] [10]. We propose a decoupled approach to solve this problem, in which we leverage dedicated algorithms to tackle the spatial, temporal and floor-contention problems separately. These algorithms are centralized, and iteratively refine the trajectory envelopes so as to remove the possibility of collisions and deadlocks, as well as those trajectories that do not satisfy the spatial and temporal constraints. De-centralized approaches to some or all of these sub-problems are possible, 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 [11]. Note also that while polynomial-time distributed algorithms can be used to guarantee safe navigation (see, $e.g.$, [12]), these cannot enforce adherence to temporal constraints like deadlines.

In order to enforce the absence of possible collisions and deadlocks in the trajectory envelopes of the fleet, several strategies are possible. One is to refine only the spatial envelopes of spatially and temporally intersecting trajectory envelopes so as to eliminate condition (4). Another possibility is to add temporal constraints that eliminate condition (5). The third option is to perform one or both refinements, depending on some particular heuristic indicating the impact of the refinement on the feasible trajectories. In this paper, we explore the second option. Specifically, we assume that an initial set of trajectory envelopes $\mathcal{E}$ is available, that the spatial envelopes $\mathcal{S}$ are fixed for each vehicle, and then refine the temporal envelopes by adding constraints $\mathcal{T}_a$ to $\mathcal{T}$ to obtain a feasible set $\mathcal{E}_a = (\mathcal{S}, \mathcal{T} \cup \mathcal{T}_a)$ of trajectory envelopes.

Definition 5: Given the models of N vehicles and $\mathcal{E}$, a *trajectory scheduling problem* consists of finding $\mathcal{T}_a$ such that $\mathcal{E}_a$ is feasible.

Or in other words, the trajectory scheduling algorithm resolves concurrent use of floor space by altering *when* different vehicles cannot occupy spatially overlapping polyhedra. Crucially, a solution to the trajectory scheduling problem does not represent a commitment to a particular assignment of times to $\boldsymbol{t}$, rather a set of additional constraints $\mathcal{T}_a$ on these times such that collisions are avoided.

Our approach is reminiscent of time scaling algorithms for multi-robot systems [13], [14], in which the temporal profiles of nominal trajectories are adjusted in order to avoid collisions and to account for dynamic properties. Our approach is similar in that it focuses on temporal adjustments, however, our approach also guarantees adherence to externally imposed temporal constraints, and does not commit to a specific path until execution time. This allows vehicle tracking controllers to deviate both in time and in space from the nominal trajectory within the bounds prescribed by the trajectory envelopes.

III. Defining Spatial Envelopes

It is often the case in industrial AGV fleet deployment scenarios that paths are given. This is the case, for instance, in several warehouse and port automation domains, where autonomous forklifts are constrained to navigate along predefined paths, as well as in underground mines, where fixed trajectories are recorded along tunnels [15]. If paths are not completely pre-defined, it is likely that navigable areas are given, and vehicle paths are planned within them.

Delimiting areas of the floor that are navigable is effectively one way to specify the spatial envelope $\mathcal{S}$ of the fleet of vehicles. $\mathcal{S}$ can be given, or it can be calculated starting from an initial reference path for each pair of destinations in the map. In this paper, we employ the latter approach. Specifically, we use a lattice-based path planner to compute optimal or highly-optimized paths between destinations (where each node of the lattice represents a pose of the vehicle in the form (x, y, θ)). The cost function is based on the distance between nodes (along the edges) of the lattice, scaled by a cost factor that penalizes backwards and turning motions. These paths can be seen as an initial, very tight collection of polyhedra, which is then relaxed so as to obtain a larger spatial envelope for each vehicle.

[2]A deadlock, in our case, is a situation in which two or more vehicles are each waiting for the other to exit a polygon, and thus neither ever does.

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

The planner uses a set of pre-defined, kinematically feasible motion primitives, which are repeatedly applied to obtain a directed graph which covers the state space. The graph is then explored using A^* [16], or one of its most efficient anytime versions, ARA^* [17], which can provide provable bounds on sub-optimality. Effective heuristic functions [18], as well as off-line computations for collision detection, are employed to speed up the exploration of the lattice. Our approach is inspired by existing lattice-based path planners [19], [20], successfully used in real world applications. All paths are generated such that there exists a time profile that yields feasible trajectories (see *Definition* 1).

Once a reference path is obtained for each vehicle, its spatial envelope is calculated by sampling it with a given $\Delta\sigma$ (inversely proportional to the curvature of the path) and calculating the polyhedron $\mathcal{S}_i^{(j)}$, enclosing the sampled point (x_i, y_i), by accounting for possible intersections of a bounding box of the vehicle and obstacles from the environment. The detailed description of the heuristics we use is outside the scope of this paper.

IV. Defining Temporal Envelopes

As stated by *Definition* 5, the trajectory scheduling problem consists of finding appropriate temporal constraints such that no collisions or deadlocks occur ($i.e.$, the set of envelopes $\mathcal{E}_a$ is feasible). The spatial envelopes $\mathcal{S}$ of the vehicles are either given or obtained as a relaxation of reference paths as described in Section III, and these constraints are not subject to change. Conversely, the process of determining the temporal envelopes $\mathcal{T} \cup \mathcal{T}_a$ is split into two parts.

First, an initial temporal envelope $\mathcal{T}^{(j)}$ is calculated for each vehicle. These envelopes contain the constraints (2) and (3). The resulting set of constraints $\mathcal{T}$ constitutes a STP which admits, by construction, at least one solution (assignment of $\boldsymbol{t}$). Note, however, that this solution may not be conflict-free, because there is no constraint in $\mathcal{T}$ that disallows $\mathcal{C} \neq \emptyset$. A second step is thus necessary, which (through scheduling) determines constraints $\mathcal{T}_a$ that lead to refined temporal envelopes $\mathcal{T} \cup \mathcal{T}_a$. In the following paragraphs, we detail both these steps.

A. Initial Temporal Envelope Definition

Here, we describe our heuristics for forming the lower and upper bounds in (2) and (3). We assume that each spatial envelope $\mathcal{S}^{(j)}$ has an associated reference path. We focus on vehicle j, hence the superscript (j) is omitted.

Let L_i denote the length of the path segment fully contained within the projection of polyhedron i on (x, y) (which we call polygon i, see Fig. 1). Furthermore, let $L_{i-1,i}$ denote the length of the path segment fully contained within the intersection of polygons $i-1$ and i. The recursion

$$s_i^{\text{fast}} = e_{i-1}^{\text{fast}} - \frac{L_{i-1,i}}{\overline{v}}, \quad e_i^{\text{fast}} = s_i^{\text{fast}} + \frac{L_i}{\overline{v}}$$
$$s_i^{\text{slow}} = e_{i-1}^{\text{slow}} - \frac{L_{i-1,i}}{\underline{v}}, \quad e_i^{\text{slow}} = s_i^{\text{slow}} + \frac{L_i}{\underline{v}}$$

defines $[s_i^{\text{fast}}, e_i^{\text{fast}}]$ ($[s_i^{\text{slow}}, e_i^{\text{slow}}]$) as the earliest (latest) possible entry and exit times for polygon i. For $i = 1$ we assume $s_i^{\text{slow}} = s_i^{\text{fast}} = 0$. The lower and upper bounds of (2) and (3) are thus obtained as

$$\ell_i = e_i^{\text{fast}} - s_i^{\text{slow}}, \quad u_i = e_i^{\text{slow}} - s_i^{\text{fast}},$$
$$\ell_{i,i+1} = e_i^{\text{fast}} - s_{i+1}^{\text{slow}}, \quad u_{i,i+1} = e_i^{\text{slow}} - s_{i+1}^{\text{fast}}.$$

B. Refining Temporal Envelopes through Scheduling

In order to prune out of $\mathcal{E}$ those trajectories that lead to collisions, we must add more temporal constraints to $\mathcal{T}$ so as to eliminate overlapping temporal intervals corresponding to overlapping spatial constraints.

Function `ScheduleTrajectories(`$\mathcal{E}$`)`: success or failure

```
1  static T_a = ∅
2  form C
3  while C ≠ ∅ do
4      c ← Choose(C, H_c) // c ∈ C is a conflict (Definition 3)
5      R_c = { s_i^(j) ≥ e_k^(m), e_i^(j) ≤ s_k^(m) }
6      while R_c ≠ ∅ do
7          r ← Choose(R_c, H_r)
8          R_c ← R_c \ r // remove constraint r from R_c
9          T_a ← T_a ∪ r // add constraint r to STP
10         if STP is consistent then
11             if ScheduleTrajectories(E_a) = failure then
12                 T_a ← T_a \ r
13             else return success
14         else T_a ← T_a \ r
15     return failure
16 return success
```

Finding a set of additional constraints that make $\mathcal{E}_a$ feasible can itself be cast as a CSP. The variables of this CSP are *conflict sets*, $i.e.$, pairs of polygons that intersect and whose associated temporal variables may overlap (see *Definition* 3). The values of these variables are temporal constraints that eliminate this temporal overlap (*resolving constraints*). Algorithm `ScheduleTrajectories()` solves the trajectory scheduling problem with a standard CSP backtracking search. It is inspired by the Earliest Start Time Approach (ESTA) precedence-constraint posting algorithm [21] for resource scheduling. The algorithm starts by collecting all pairs of conflict sets (line 2). The assessment of condition (5) (possible temporal overlap) is performed by comparing the Earliest Time (ET) solutions of the STP. In other words,

$$\left[s_i^{(j)}, e_i^{(j)}\right] \cap \left[s_k^{(m)}, e_k^{(m)}\right] \neq \emptyset \text{ if}$$
$$\max\left(\underline{s}_i^{(j)}, \underline{s}_k^{(m)}\right) \leq \min\left(\underline{e}_i^{(j)}, \underline{e}_k^{(m)}\right), \qquad (6)$$

where the lower bound of the time points are determined by the constraints $\mathcal{T} \cup \mathcal{T}_a$ in the STP.

As usual in CSP search, the variables (conflict sets) are ordered according to a most-constrained-first variable ordering heuristic. In our case, $\mathbb{H}_c$ gives preference to pairs of polygons that are spatially closer to other conflicting pairs (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 (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. Note that since conflict sets are pairs of indexes, there are only two ways to resolve the temporal overlap, namely imposing that the end time of polyhedron $\mathcal{S}_i^{(j)}$ is constrained to occur before the start time of polyhedron $\mathcal{S}_k^{(m)}$, or vice-versa. Again as is common practice in CSP search, the value (resolving constraint) to attempt first is chosen (line 7) according to a least constraining value ordering heuristic. $\mathbb{H}_r$ in our case leverages temporal slack to choose the ordering that least affects the temporal flexibility of the time points [21]. 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.

The `ScheduleTrajectories()` algorithm is complete, as it performs a systematic search in the space of possible sequencing constraints for conflicting polygons. Thus, it will return a set of deadlock- and conflict-free trajectory envelopes if and only if such a set exists.

V. Obtaining Discrete Trajectory Envelopes

Once trajectory scheduling has taken place, we are left with a fully propagated and consistent STP, $i.e.$, one in which the bounds of all time points $s_i^{(j)}$ and $e_i^{(j)}$ have been updated to reflect the constraints $\mathcal{T} \cup \mathcal{T}_a$. From this STP, we can obtain the ET solution $\boldsymbol{t}_1$, that is, the solution in which all variables are assigned to their lower bounds $\underline{s}_i^{(j)}$. This solution satisfies all constraints $\mathcal{T} \cup \mathcal{T}_a$. Since it is the ET solution, it is therefore also conflict-free. Note that this solution defines the fastest possible trajectory for all vehicles that is conflict free.

We can obtain a further solution $\boldsymbol{t}_2$ by propagation, $i.e.$, by adding the constraint $s_1^{(j)} \geq \underline{s}_1^{(j)} + \Delta t$, where $\underline{s}_1^{(j)}$ is the value of $s_1^{(j)}$ in the ET solution of the STP. We can obtain a total of P solutions by repeating this procedure P times. Although the ET solution satisfies all constraints in $\mathcal{T} \cup \mathcal{T}_a$, the addition of the constraints above may have generated a conflict that couldn't have been detected with the ET assessment (6) performed in the previous invocation of `ScheduleTrajectories()`. This is because conflicts are identified using the ET solutions of the STP (6). Hence, the scheduler must be re-invoked $P-1$ times to check if the addition of the new constraints entails a new conflict.

In practice, re-scheduling rarely finds new conflicts. This is because a previously unseen conflict set appears only if some polyhedron is subject to "tighter" temporal constraints than others ($e.g.$, externally imposed slow-down areas). The overhead of checking for conflicts is $O(|S|^2)$, as conflicts involve pairs of polygons. Since Floyd-Warshall can be run incrementally in $\Theta((2\,|\mathcal{S}|)^2)$ [7], we can typically obtain P discrete trajectory envelopes for all vehicles with a quadratic computational overhead.

We leverage these properties of the STP to obtain a set of P discrete trajectory envelopes (see Section II-A) for each vehicle j. Note that as long as each vehicle follows the p-th discrete trajectory envelope in its set, no collisions will occur. However, if one vehicle's controller must deviate from the p-th discrete trajectory and selects to follow the p'-th, then all vehicles must adhere to their p'-th discrete trajectory envelope. For this reason, we centralize the allocation of discrete trajectory envelopes to vehicles, as described below.

A. Selecting the Current Discrete Trajectory Envelope

Once P discrete trajectory envelopes are sampled out of $\mathcal{E}_a$, it is the task of a central *trajectory envelope selector* to dispatch these envelopes to the vehicle controllers. In addition, the trajectory envelope selector also instructs all vehicles to follow a reference trajectory $\boldsymbol{p}^r(\sigma)$ that lies within their discrete trajectory envelope $p_e \in \{1, \ldots, P\}$.

The selection of the current discrete trajectory envelope p_e is revised as vehicles execute their missions. As we will see, revision of the current p_e can be performed based on information that is fed back to the centralized trajectory envelope selector from individual vehicle controllers. In essence, the controller for vehicle j feeds back $J_p^{(j)}$, which is a measure of the *tracking performance* of vehicle j within the discrete trajectory envelope p. If a discrete trajectory envelope p for vehicle j is found to be infeasible by the controller due to the current state of the vehicle, $J_p^{(j)} = \infty$. Based on these measures, the trajectory envelope selector can periodically revise the current trajectory envelope index and inform all vehicle controllers of the new global choice. More formally, the trajectory envelope selector computes

$$p_e \in \underset{p \in \{1,\ldots,P\}}{\arg\min}\; g(J_p^{(1)}, \ldots, J_p^{(N)}).$$

In this way, by analyzing the tracking performance of all vehicles, the trajectory envelope selector can make informed decisions as to which index p_e represents the globally best set of discrete trajectories to be followed by all vehicles.

The particular form of the function g which is minimized is the topic of future work — in our present work, we assume that the fastest set of feasible trajectories is selected.

VI. Tracking Controller

In this section we present the tracking controller that is used in combination with the trajectory scheduler and trajectory envelope selector. We focus on the controller for the j-th vehicle and drop the superscript (j) for compactness.

Recall that an underlying assumption of our trajectory scheduling algorithm is that the reference point of the vehicle stays within $\langle \mathcal{S}_{I_1}^{(j)}, \mathcal{S}_{I_2}^{(j)}, \ldots, \mathcal{S}_{I_D}^{(j)} \rangle$. In order to explicitly account for such state constraints, as well as for constraints on the control variables, we use an embedded optimization based approach. Compared to classical tracking controllers (which do not account explicitly for constraints $e.g.$, see [6]), this approach is computationally more expensive. However, polyhedral constraints (such as those used to define trajectory envelopes here) can be handled very efficiently. Depending

on formulation and underlying assumptions, it is possible to compute control actions in the millisecond (or even microsecond) range.

Model Predictive Control (MPC) is one of the most successful embedded optimization schemes. It has been used in a wide variety of industrial applications [22]. In fields that involve linear and/or rather slow systems, like the ones usually encountered in the chemical process industry, MPC is considered a mature technique. The application of MPC to fields that involve highly nonlinear, hybrid, or very fast processes is still an active research topic ($e.g.$, the use of MPC for walking motion generation is discussed in [23]). The application of MPC in the context of tracking a reference trajectory using a nonholonomic vehicle has been discussed in [24], [25], [26], [27], [28].

The basic idea behind MPC is to use a model of the process of interest (which in our case is the evolution of the state of a car-like vehicle), and compute a sequence of control actions (at each control sampling time) that optimize a given objective, while satisfying input and state constraints, over a given time interval (referred to as *preview horizon*). After the state is transferred as a result of the first control action (in the sequence), a new optimization problem is formulated and solved (giving a new sequence of actions, from which again only the first one is executed).

As discussed in Section V, each vehicle is assigned P sequences of indexes $\{I^p\}$, $p \in \{1, \ldots, P\}$ (I^p defines the p-th discrete trajectory envelope). It is also given the current index p_e of the discrete trajectory envelope that should be followed. The tracking controller has two purposes. One is to compute the control actions for the vehicle that achieve a close enough execution compared to the reference trajectory. associated to the p_e-th discrete trajectory envelope. Its second purpose is to provide a measure of how well the vehicle can perform on *other* discrete trajectory envelopes. As mentioned in Section V-A, a centralized trajectory envelope selector takes this information into account when choosing p_e. Note that both the trajectory scheduler and the MPC perform redundant computation – the former pre-computes trajectory envelopes which are all guaranteed to be conflict-free, the latter evaluates them to facilitate a global selection of "good" trajectory envelopes for *all* vehicles, $i.e.$, an index p_e that maximizes the tracking performance of all vehicles.

A. Linear MPC Formulation

The tracking controller formulation is presented using the reference trajectory $\boldsymbol{p}^r(\sigma)$ associated to the p-th discrete trajectory envelope (of the j-th vehicle). By using the differential flatness property of (1), we can obtain [6] associated reference state and control trajectories $\boldsymbol{q}^r(t)$, $\boldsymbol{v}^r(t)$ that (by construction) satisfy $\dot{\boldsymbol{q}}^r(t) = \boldsymbol{f}(\boldsymbol{q}^r(t), \boldsymbol{v}^r(t))$.

With the assumption that the vehicle starts "close" to the reference trajectory, we linearize (1) along $(\boldsymbol{q}^r(t), \boldsymbol{v}^r(t))$, and discretize using forward difference to obtain the following discrete-time, time-varying linear dynamical system

$$\Delta \boldsymbol{q}_{k+1} = \boldsymbol{A}_k \Delta \boldsymbol{q}_k + \boldsymbol{B}_k \Delta \boldsymbol{v}_k, \quad k = 0, \ldots, D-1, \quad (7)$$

where $\Delta \boldsymbol{q}_k = \boldsymbol{q}_k - \boldsymbol{q}_k^r$, $\Delta \boldsymbol{v}_k = \boldsymbol{v}_k - \boldsymbol{v}_k^r$ and

$$\underbrace{\begin{bmatrix} 1 & 0 & -\sin(\theta_k^r) v_k^r \Delta t & 0 \\ 0 & 1 & \cos(\theta_k^r) v_k^r \Delta t & 0 \\ 0 & 0 & 1 & \frac{v_k^r \Delta t}{l \cos(\phi_k^r)^2} \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{A}_k}, \underbrace{\begin{bmatrix} \cos(\theta_k^r)\Delta t & 0 \\ \sin(\theta_k^r)\Delta t & 0 \\ \frac{\Delta t \tan(\phi_k^r)}{l} & 0 \\ 0 & \Delta t \end{bmatrix}}_{\boldsymbol{B}_k},$$

with Δt being the sampling time. At each sampling time we solve the following quadratic program (QP)

$$\text{minimize } J = \sum_{k=1}^{D} \Delta \boldsymbol{q}_k^T \boldsymbol{Q}_k \Delta \boldsymbol{q}_k + \sum_{k=0}^{D-1} \Delta \boldsymbol{v}_k^T \boldsymbol{R}_k \Delta \boldsymbol{v}_k \quad (8)$$

$$\text{subject to (7), with a given } \Delta \boldsymbol{q}_0$$

$$\boldsymbol{q}_k \in \mathcal{S}_{I_k^p}, \quad \boldsymbol{v}_{k-1} \in \mathcal{V}_{k-1}, \quad k = 1, \ldots, D,$$

with decision variables[4] the $\Delta \boldsymbol{q}_k$'s and $\Delta \boldsymbol{v}_k$'s. D is the number of sampling times in the preview window, $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ are assumed to be positive-definite matrices that penalize state and control deviation. The $\mathcal{V}_k$'s are assumed to be compact sets that contain the origin in its interior and p is given. Note that the sequence $\langle \mathcal{S}_{I_k} \rangle$ can be used to define regions for which the linear approximation is "good enough".

The optimal trajectory envelope for vehicle j is defined as

$$p^{\star(j)} = \underset{p \in \{1,\ldots,P\}}{\arg\min} \left(J_p^{\star(j)} \right),$$

where $J_p^{\star(j)}$ is the value of J at the solution (for vehicle j).

Overall, each vehicle is given P trajectory envelopes within which to follow a reference trajectory. Each of these trajectory envelopes leads to one QP, for which the solution is found. We note again that vehicle j does not necessarily execute $p^{\star(j)}$, rather the solution of the p_e-th QP is used to provide control inputs for the vehicle (so as to ensure that each vehicle acts consistently with respect to the other vehicles). A measure of how well the vehicle would perform on all trajectory envelopes is used to inform the centralized trajectory envelope selector, whose task it is to choose which of the P trajectory envelopes would best suit the vehicle given its current state and the state of other vehicles.

VII. Evaluation

We now present an experimental validation of our approach, focusing on the performance of trajectory scheduling and on the use of discrete trajectory envelopes for control. All test runs have been performed in simulation on an ordinary laptop computer, and the same kinematic and geometric models were employed for all vehicles, namely those of a Linde H50D forklift (inset in Fig. 2). All experiments employ minimum and maximum speeds $(\underline{v}, \overline{v}) = (2, 10)\, m/s$ to obtain the initial temporal envelopes $\mathcal{T}$.

[4]For simplicity of notation we have assumed that the indexes (k) of reference states $\boldsymbol{q}^r$ and control inputs $\boldsymbol{v}^r$ in a preview window always start from 0. Hence, $\Delta \boldsymbol{q}_0$ always denotes (an estimation of) the current state (even though (7) is time varying).

The lattice-based path planner used to compute the initial reference path for spatial envelope computation employs a grid resolution of 0.2 meters, 16 angles for θ (equally spaced in $[-\pi, \pi]$), and each vertex is connected to 15 others through pre-calculated, kinematically feasible motion primitives.

A. 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 Fig. 2), where we deployed 7 identical vehicles with pre-assigned start and destination poses.

Reference paths were generated using ARA^* with a 5 second cut-off time. A total of 140 polyhedra were computed for the 7 vehicles. The `ScheduleTrajectories()` algorithm identified three groups of conflicting polyhedra (shaded in Fig. 2). $\mathcal{T}_a$, the solution of the scheduling CSP, consisted of 13 temporal constraints.

Extracting a specific trajectory for execution other than the ET trajectory took about 250 milliseconds (no further conflicts were found). The total time required to generate 10 discrete trajectory envelopes was less than 40 seconds: less than 5 seconds for the path planning, and 34 seconds for scheduling and discrete trajectory envelope generation.

B. 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 width and length of 50 meters, we pre-defined 80 poses, 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 or 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 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. 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 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).

Again focusing on trajectory scheduling efficiency, 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 Fig. 3 (the percentage of satisfiable problems is shown on top of each set of results.) As expected, scheduling time grows exponentially

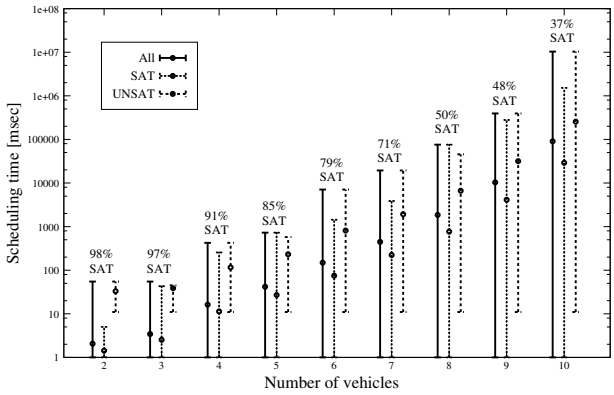

Fig. 3. Quantitative evaluation of the trajectory scheduler.

with the number of vehicles involved. This time does not include discrete trajectory envelope generation (all conflicts are found the first time scheduling occurs, see Section V).

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 time. Moreover, all vehicles are constrained to start moving at the same time, as all starting polyhedra 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, the calculation of each set of discrete trajectory envelopes never takes more than 50 milliseconds (the most challenging problem of our benchmark contains 94 polyhedra).

Considering $D = 30$ sampling times in the preview window, a condensed reformulation of QP (8) ($i.e.$, with the equality constraints in (7) eliminated) can typically be solved in under 4 ms (plus 1 ms for condensing). Alternatively, we can leverage the sparsity pattern of the QP's original formulation to obtain even better performance [29], [23].

VIII. Conclusions

In this paper we have presented a constraint-based approach to multiple vehicle coordination based on the notion of least commitment: vehicle trajectories are progressively refined by several constraint solvers to obtain multiple sets of constraints on trajectories which guarantee conflict-free execution. These sets of constraints, which we have called *trajectory envelopes*, represent alternative execution patterns for the vehicle controllers to track. All solvers perform redundant computation: vehicle controllers evaluate alternative trajectory envelopes and inform a centralized trajectory selector of their current tracking performance; scheduling computes these alternative trajectory envelopes and informs vehicles about which one should be adhered to for execution.

The primary advantage of this approach is that it allows to take into account requirements at all levels of decision making. This is essential in industrial applications, where reference paths and temporal constraints on tasks may be completely or partially specified (and modification of these

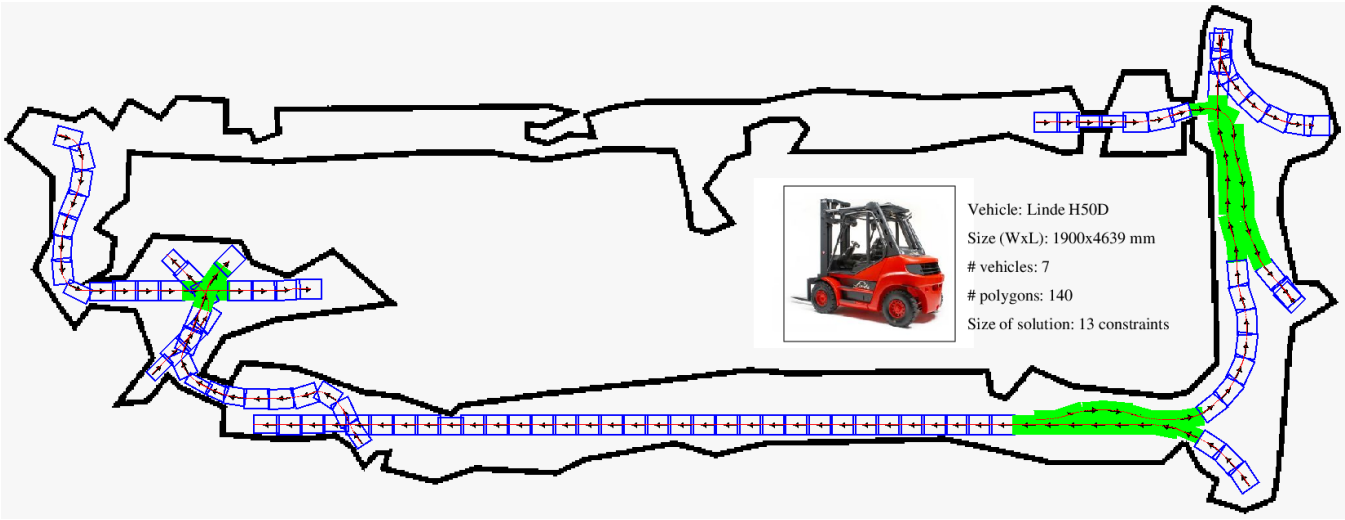

Fig. 2. A solution to a mission planning problem involving seven vehicles in an underground mine. Polyhedra involved in conflicts during trajectory scheduling are shaded.

requirements is undesirable). We have shown through a brief experimental evaluation that our approach scales well to realistically sized and structured scenarios.

The approach described in this paper provides a means to partially handle uncertainty in the position of vehicles within given constraints on time and space. We do not handle other types of uncertainty, such as perceptual uncertainty — this will be one of the directions of future work.

REFERENCES

[1] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[2] R. Luna and K. Bekris, "Efficient and complete centralized multi-robot path planning," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

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

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

[5] V. Desaraju and J. How, "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)*, 2011.

[6] A. de Luca, G. Oriolo, and C. Samson, "Feedback Control of a Non-holonomic Car-like Robot," in *Robot Motion Planning and Control*, J.-P. Laumond, Ed., 1998, ch. 4, pp. 171–253.

[7] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, no. 1-3, pp. 61–95, 1991.

[8] R. W. Floyd, "Algorithm 97: Shortest path," *Communication of the ACM*, vol. 5, pp. 345–348, June 1962.

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

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

[11] B. Faltings, "Distributed constraint programming," in *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh, Eds. Elsevier, 2006, pp. 699–729.

[12] K. E. Bekris, D. K. Grady, M. Moll, and L. E. Kavraki, "Safe distributed motion coordination for second-order systems with different planning cycles," *International Journal of Robotics Research (IJRR)*, vol. 31, no. 2, 2012.

[13] I. Komlósi and B. Kiss, "Motion planning for multiple mobile robots using time-scaling," in *Mobile Robots – Control Architectures, Bio-Interfacing, Navigation, Multi Robot Motion Planning and Operator Training*, J. Będkowski, Ed. InTech, 2011.

[14] S. Moon and S. Ahmad, "Time scaling of cooperative multirobot trajectories," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 21, no. 4, pp. 900–908, 1991.

[15] J. Larsson, "Unmanned operation of load-haul-dump vehicles in mining environments," Ph.D. dissertation, Örebro University, School of Science and Technology, 2011.

[16] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

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

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

[19] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[20] C. Urmson, J. Anhalt, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[21] A. Cesta, A. Oddi, and S. F. Smith, "A constraint-based method for project scheduling with time windows," *Journal of Heuristics*, vol. 8, no. 1, pp. 109–136, January 2002.

[22] S. Qin and T. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, pp. 733–764, 2003.

[23] D. Dimitrov, A. Sherikov, and P.-B. Wieber, "A sparse model predictive control formulation for walking motion generation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011, pp. 2292–2299.

[24] D. Gu and H. Hu, "Receding Horizon Tracking Control of Wheeled Mobile Robots," *IEEE Trans. on Control Systems Technology*, vol. 14, no. 4, pp. 743–749, 2006.

[25] A. Divelbiss and J. Wen, "Trajectory Tracking Control of a Car-Trailer System," *IEEE Trans. on Control Systems Technology*, vol. 5, no. 3, pp. 269–278, 1997.

[26] F. Kühne, W. Lages, and J. Gomes, "Point Stabilization of Mobile Robots with Nonlinear Model Predictive Control," in *IEEE Internat. Conf. on Mechatronics and Automation*, 2005, pp. 1163–1168.

[27] C. Liu, W. Chen, and J. Andrews, "Experimental Tests of Autonomous Ground Vehicles with Preview," *International Journal of Automation and Computing*, vol. 7, no. 3, pp. 342–348, 2010.

[28] Y. Zhu and U. Özgüner, "Constrained Model Predictive Control for Nonholonomic Vehicle Regulation Problem," in *17th World Congress IFAC*, 2008, pp. 9552–9557.

[29] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, no. 2, pp. 267–278, 2010.