A Lattice-Based Approach to Multi-Robot Motion Planning for Non-Holonomic Vehicles

Marcello Cirillo[1], Tansel Uras[2] and Sven Koenig[2]

***Abstract*— Coordinating fleets of autonomous, non-holonomic vehicles is paramount to many industrial applications. While there exists solutions to efficiently calculate trajectories for individual vehicles, an effective methodology to coordinate their motions and to avoid deadlocks is still missing. Decoupled approaches, where motions are calculated independently for each vehicle and then centrally coordinated for execution, have the means to identify deadlocks, but not to solve all of them. We present a novel approach that overcomes this limitation and that can be used to complement the deficiencies of decoupled solutions with centralized coordination. Here, we formally define an extension of the framework of lattice-based motion planning to multi-robot systems and we validate it experimentally. Our approach can jointly plan for multiple vehicles and it generates kinematically feasible and deadlock-free motions.**

I. Introduction

Automatically Guided Vehicles (AGVs) have been deployed in large numbers for industrial intra-logistics tasks. The industry standard approach to the management of fleets of AGVs still relies on fixed paths [1] and manually specified traffic rules. This approach presents two drawbacks: First, traffic rules cannot guarantee deadlock-free coordination, and, second, even small modifications to the environment require defining new paths and manually updating the traffic rules, both of which are costly and inflexible procedures.

Several approaches have been proposed to quickly calculate drivable trajectories for non-holonomic vehicles deployed in the real world [2], [3]. These methods can be used also in industrial environments, but they are applicable only to individual vehicles. Multi-robot motion planning is to date an open research problem. Even under very simplified assumptions, such as considering robots which move on a grid in discrete time steps, state-of-the-art optimal algorithms can handle only a limited number of robots on a relatively small map. On the other hand, there exists efficient non-optimal algorithms which can coordinate a large number of robots [4], but they do not offer guarantees on the quality of the resulting paths nor on termination time.

Recently, a new solution has been proposed for multi-robot coordination in industrial environments [5], where kinematically feasible motions are calculated for each vehicle independently. These motions are subsequently passed to a coordinator which assigns time profiles to the paths, resulting in coordinated trajectories. The approach does not guarantee global optimality, but it allows for motions to be calculated and scheduled online, rather than being manually defined. More importantly, as the coordinator's algorithm is complete, deadlocks can be immediately identified by coordination failures. The main drawback of the approach is that it does not allow for spatial backtracking: If coordination fails, it is not possible to generate new motions which would break the deadlock, as they can only be calculated independently. An example of a problem which cannot be temporally coordinated can be seen in Fig. 4(a): Two vehicles at a crossing need to swap places. Their individual motions will completely overlap, thus preventing the coordinator from finding collision-free trajectories to dispatch them to their respective destinations. Here, motion planning must consider the two vehicles jointly to obtain a schedulable solution.

In this paper, we present a new lattice-based multi-robot motion planner for non-holonomic vehicles which calculates kinematically feasible motions that are guaranteed to be schedulable. The planner supports different search algorithms to explore the state space, namely A^* and ARA^*, thus offering guarantees on the quality of the solutions. It can complement the approach described above: It can be invoked whenever the central coordinator fails to find valid trajectories, focusing on those vehicles which caused an unsolvable deadlock because of spatial overlaps.

Our main contribution is the formal definition of this novel framework for lattice-based multi-robot motion planning and the analysis of its formal properties. We also present techniques to speed up the computation of the plan, and we validate our approach both in simulation and with real robots.

II. Related Work

Motion planning under differential constraints has been extensively studied in the past decades [6]. Different families of sampling-based methods are currently widely used: Probabilistic Roadmaps (PRMs) [7], Rapidly-exploring Random Trees (RRTs) [8] and lattice-based motion planners [9], all of which are effective in high-dimensional configuration spaces. PRMs have two major drawbacks: First, before running the algorithm, several parameters must be selected (e.g., the duration of the learning phase); and, second, a new roadmap has to be built every time the environment is subject to substantial changes. After their initial introduction [10], RRTs have been extensively studied, and many extensions to the original algorithm have been proposed [11], [12]. RRTs do not guarantee convergence (termination is usually implemented with a timeout) and, unless the space is analyzed beforehand, they cannot verify whether a problem offers

[1]Marcello Cirillo is with the Center for Applied Autonomous Sensor Systems, Örebro University, Sweden marcello.cirillo@oru.se

[2]Tansel Uras and Sven Koenig are with the Department of Computer Science, University of Southern California, USA {turas,skoenig}@usc.edu

no solution. Lattice-based motion planners combine the strengths of the previous approaches with classical AI graph-search algorithms, such as A^*, ARA^* [13] and D^*Lite [14]. Differential constraints are incorporated into the state space by means of pre-computed motion primitives which trap the motions onto a regular lattice. The state space is then explored using efficient graph-search algorithms. The planner presented here belongs to the lattice-based family, here extended to solve multi-robot motion planning problems.

Multi-agent path planning is a very active research area. A considerable part of the work within it focuses on the coordination of large teams of robots which are not subject to differential constraints. Centralized algorithms such as M^* [15], an extension of A^* to multi-robot systems, and "Push and Swap" [4], a computationally efficient and complete method for multi-robot path planning, are examples of promising results in this direction. A partially decentralized approach has been presented in [16], where agents still move on a grid, and an overall optimal road map is coupled with local motions calculated for each robot independently. By contrast, our algorithm can intrinsically take into account differential constraints thanks to its lattice-based nature. Approaches which can consider non-holonomic constraints typically rely on de-centralized solutions. For instance, the assignment of priority levels to robots can guarantee fast results. This can be seen as an improved version of hand-coded traffic rules, but cannot ensure deadlock-free situations [17]. Desaraju and How [18] further extend the idea of prioritized path planning, by substituting priority levels with a merit based token passed among agents. Our algorithm is centralized and complete, but could potentially present scalability issues. However, we can use an external coordinator to identify the smallest groups of robots whose individual motions are not schedulable [19], and consider only those for multi-robot motion planning. Closely related to the problem addressed in this paper are the control of formations of non-holonomic robots [20] and motion planning in the presence of dynamic obstacles [21]. Here, instead, we focus on problems where all robots are controllable and have different goals.

III. Our Approach: The Single-Robot Case

Given a model of vehicle maneuverability, the intuition behind lattice-based motion planning is to sample the state space in a regular fashion and to constrain the motions of the vehicle to a lattice graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, that is, a graph embedded in a Euclidean space $\mathbb{R}^n$ which forms a regular tiling [22]. Each vertex $v \in \mathcal{V}$ represents a valid pose, or configuration, of the vehicle, while each edge $e \in \mathcal{E}$ encodes a motion which respects the non-holonomic constraints of the vehicle. Here, since we aim at generating motions to which a temporal profile will be attached by a central coordinator, we focus only on the kinematic constraints.

Definition 1. *A model* m *of a vehicle is fully specified by the tuple* $\langle d, \Theta, \Phi, r, P, g \rangle$*, where* d *encodes the geometric dimensions of the vehicle,* Θ *a finite set of orientations,* Φ *a finite set of steering angles,* r *the resolution of an* $\mathbb{R}^2$ *regular grid,* P *a finite set of allowed motions which respect the kinematic constraints of the vehicle and* g *a cost function.*

Definition 2. *A valid* configuration *for a vehicle model* $\mathrm{m} = \langle d_m, \Theta_m, \Phi_m, r_m, P_m, g_m \rangle$ *is a four-dimensional vector* $c = \langle x, y, \theta, \phi \rangle$*, where* (x, y) *lies on a grid of resolution* r_m*,* $\theta \in \Theta_m$ *and* $\phi \in \Phi_m$*.*

P_m is the set of motion primitives of vehicle model m, which captures the mobility of the vehicle while intrinsically taking into account its kinematic constraints. Under the assumption of even terrain, we can design P_m to be position-invariant.[1] P_m can be partitioned into subsets $P_{m,\theta,\phi}$, such that $P_m = \bigcup_{\theta \in \Theta_m, \phi \in \Phi_m} P_{m,\theta,\phi}$. In free space, all primitives of each $P_{m,\theta,\phi}$ can be applied to any valid $c = \langle x, y, \theta, \phi \rangle$. P_m is calculated by using a *boundary value problem* (BVP) solver to connect a set of initial configurations $c = \langle 0, 0, \theta, \phi \rangle$ to all neighboring states in a discrete, bounded neighborhood ρ in free space. The BVP solver guarantees that the motions respect the kinematic constraints of the vehicle, while the position-invariant property ensures that the primitives are translatable to other configurations. P_m can then be reduced for efficiency using the techniques described in [23], by removing those primitives that can be decomposed into other primitives in P_m, without affecting the reachability of the configuration space of the robot in free space. The full algorithm is described in Procedure `PrimGen`. The procedure can be simplified by imposing an 8-axis symmetry on the primitives, so as to calculate P_m only for subsets of Θ_m and Φ_m, and reconstruct the remaining primitives by applying simple symmetries. The generation and reduction processes can be performed offline. Finally, a cost $g_m(p)$ is associated with each $p \in P_m$. In our implementation, $g_m(p)$ is calculated by multiplying the distance covered by p by a cost factor which penalizes backwards and turning motions.

A planning problem for a single vehicle is fully specified by the tuple $\langle m, c_s, c_g, map \rangle$, where c_s is the start configuration of the vehicle, c_g its goal configuration, and *map* an occupancy map of the environment with resolution r_m in which all known obstacles are represented. A *valid solution* to the problem is a sequence of collision-free primitives $\pi = (p_0, \ldots, p_n)$ connecting c_s to c_g. Given the set Π of all valid solutions to a problem, an optimal solution π_{opt} is the one with minimum cost:

$$\pi_{opt} = \operatorname*{argmin}_{\pi_i \in \Pi} \sum_{p_j \in \pi_i} g_m(p_j)$$

Starting from c_s, the state space can be explored using efficient graph-search algorithms. In our case, we rely on A^* or one of its most efficient anytime versions, ARA^* [13], which can provide provable bounds on sub-optimality.

A. Collision Detection

We can perform offline the brunt of the calculations necessary for collision detection. We define $s_m : P_m \mapsto 2^{\mathbb{Z}^2}$ such that $s_m(p)$ denotes the set of cells on a grid of resolution r_m traversed by the footprint of the vehicle when

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

Procedure `PrimGen` (Θ_m, Φ_m, r_m, ρ, *BVPSolver*)

$P_m = \emptyset$;
foreach $\theta \in \Theta_m$ **do**
  foreach $\phi \in \Phi_m$ **do**
    $P_{m,\theta,\phi} = \emptyset$;
    $c_0 = \langle 0, 0, \theta, \phi \rangle$;
    foreach $c_i = \langle x_i, y_i, \theta_i, \phi_i \rangle \in \rho$,
        (x_i, y_i) *on grid with res.* r_m, $c_i \neq c_0$ **do**
      $p = BVPconnect(c_0, c_i)$;
      if *success* **then**
        $P_{m,\theta,\phi} = P_{m,\theta,\phi} \cup \{p\}$;
    $P_m = P_m \cup P_{m,\theta,\phi}$;
reduce(P_m);
return P_m;

it executes primitive p in free space starting in position $(0, 0)$. We can pre-calculate and store the resulting sets for all $p \in P_m$: Detecting a collision whenever the vehicle is in a configuration $c = \langle 0, 0, \theta, \phi \rangle$ and primitive p is applied is simply a matter of checking whether all cells in the set are free in the occupancy map. As the primitives are position invariant, it is straightforward to calculate online $s_{m,x,y} : P_m \mapsto 2^{\mathbb{Z}^2}$, such that $s_{m,x,y}(p)$ denotes the set of cells swept by p when the vehicle starts its motion at (x, y), and check whether all cells in $s_{m,x,y}(p)$ are free in the occupancy map.

B. Heuristic Functions

We rely on a combination of different heuristic functions to guide the search on the lattice. In free space, we employ two admissible heuristic functions: Euclidean distance and optimum distance hash table [24]. The hash table is pre-computed offline: We run a Dijkstra algorithm from selected starting configurations with a cutoff cost and we store the optimum configuration-to-configuration costs. The hash table size is kept small by leveraging the position invariance of our primitives and the fact that we build them with an 8-axis symmetry. We also employ a third heuristic function, to be used in cluttered environments to avoid useless exploration of the lattice in blocked areas (as in the bug-trap problem). In these cases, we run a wavefront algorithm from the cell in the occupancy map which contains c_g using 8-neighbor grid connectivity. We save the distance from the goal in each cell and we use it to guide the search: The value of this heuristic function for a configuration is equal to the distance saved in the cell which contains the configuration. As this last heuristic function is not necessarily admissible for non-holonomic vehicles, we use it only in cluttered environments. The final heuristic value we use in each state is the maximum of the single heuristic functions described above.

IV. Extension to Multi-Robot Case

Single-robot motion planning is not sufficient to solve problems like the one of Fig. 4(a). To find motions that can be temporally coordinated, we need to consider the movements of the two vehicles jointly. We now define a new extension of the framework of lattice-based motion planning to multi-robot systems.

Definition 3. *A vehicle* $\mathtt{v}_i$ *is fully specified by the pair* $\langle i, m_i \rangle$*, where* i *is a unique identifier and* m_i *is the model of the vehicle.*

Note that the above definition entails that we can consider vehicles with different models and also multiple vehicles with the same model. In the implementation presented in this paper, we impose the constraint that all vehicles' models have the same resolution r.

Definition 2 still holds in the multi-robot case for each individual vehicle, but here the state space is not any longer a lattice graph. Now, each state in the state space represents a joint configuration of all vehicles:

Definition 4. *Given a set of N vehicles, a* state *is an ordered set* $C = \{c_1, \cdots, c_N\}$*, where each* $c_i \in C$ *is a valid configuration of vehicle* $\mathtt{v}_i$ *and there is no spatial overlap between any two configurations.*

Spatial overlap between configurations can be detected using the geometric information d (e.g., the vehicle's footprint), as defined for each model.

Definition 5. *A multi-robot motion planning problem is fully specified by the tuple* $\langle V, C_s, C_g, map \rangle$*, where:*

- $V = \{\mathtt{v}_1, \cdots \mathtt{v}_N\}$ *is an ordered set of vehicles;*
- $C_s = \{c_s^1, \cdots, c_s^N\}$ *is an ordered set of start configurations, such that* c_s^j *is the start configuration of* $\mathtt{v}_j$*;*
- $C_g = \{c_g^1, \cdots, c_g^N\}$ *is an ordered set of goal configurations, such that* c_g^j *is the goal configuration of* $\mathtt{v}_j$*;*
- map *is a grid map of the environment with resolution* r *in which all known obstacles are represented.*

A *valid global solution* to a multi-robot motion planning problem is an ordered set $\pi_{glob} = (\pi_1, \cdots, \pi_N)$ such that, π_i is a valid solution for vehicle $\mathtt{v}_i$ for every $1 \leq i \leq N$, and there exists at least one valid schedule that avoids collisions among vehicles. An optimal solution to the multi-robot motion planning problem is a valid global solution which minimizes the sum of the costs for all vehicles. In the following, we explain how we build the state space and search it to find a valid global solution.

A. Successor Generation

We build the state space of a multi-robot motion planning problem by moving one vehicle at a time, while considering the others as obstacles (see Procedure `SuccessorGeneration`). Given a multi-robot motion planning problem with N arbitrarily ordered vehicles, a state corresponds to a point $C = \{c_1, \cdots, c_N\}$ in the joint configuration space, where $c_i = \langle x_i, y_i, \theta_i, \phi_i \rangle$ is a valid configuration for the i-th vehicle. Individually, each vehicle $\mathtt{v}_i$ has a set P_{m_i,θ_i,ϕ_i} of applicable primitives. We generate the successors of a state $\bar{C} = \{\bar{c}_1, \cdots, \bar{c}_N\}$ by selecting each vehicle $\mathtt{v}_i$ in turn and by applying all primitives in $P_{m_i,\bar{\theta}_i,\bar{\phi}_i}$ to its configuration $\bar{c}_i$, while considering the other vehicles as obstacles. This entails that, in each successor, all individual configurations remain unchanged but one and the number of successors of $\bar{C}$ is at most $\sum_{i=1}^{N} |P_{m_i,\bar{\theta}_i,\bar{\phi}_i}|$.

Procedure `SuccessorGeneration(`C`)`

```
successors = ∅;
foreach c_i ∈ C do
    foreach c_j ∈ C, c_j ≠ c_i do
        addObstacle(c_j, map);
    foreach p ∈ P_{m_i,θ_i,φ_i} do
        c'_i = succ(c_i, p);
        if p, c'_i are collision free then
            C' = c'_i ∪ (⋃_{c_j ∈ C\c_i});
            successors = successors ∪ C';
    foreach c_j ∈ C, c_j ≠ c_i do
        removeObstacle(c_j, map);
return successors;
```

B. Exploring the State Space

The state space can be explored with any graph-search algorithm. Each vertex $\hat{v}$ corresponds to a unique state $\hat{C}$ in the joint configuration space. The cost associated with $\hat{v}$ is equal to the cost of the shortest known path from C_s to $\hat{C}$, and an optimal solution is a minimum-cost path from C_s to C_g. In our implementation, we search the state space using A^* and ARA^* [13]. Note that the single-robot case is a special case of the multi-robot motion planning problem. Here, we use the same heuristic functions described in Section III-B, and the heuristic value of a state is calculated as the sum of the heuristic values of the individual configurations (the sum of the individual admissible heuristic functions is admissible).

C. Formal Properties

In the following, we provide proofs of *completeness* and *correctness* for our algorithm under three assumptions. This means that, under the assumptions below, if a multi-robot motion planning problem admits a solution, then the planner will find it, and any valid solution returned will be schedulable. These assumptions account for the continuous nature of space and the fact that the algorithm does not allow for concurrent movements during search: (1) The map used by the motion planner is appropriately discretized given the operational conditions of the vehicles; (2) the set of primitives for each vehicle model is chosen appropriately; and (3) parallel motions are never *required* to solve a multi-robot motion planning problem.[2] Under these assumptions:

Lemma 1. (*Completeness) Given a multi-robot motion planning problem the motion planner is complete.*

Proof. Completeness follows from the fact that the algorithm performs a systematic search. □

Lemma 2. (*Correctness) If the algorithm finds a joint plan from C_s to C_g, the plan is guaranteed to be schedulable.*

Proof. A joint plan extracted from the path in the state space connecting C_s to C_g corresponds already to a valid schedule. In the path connecting the two states, each edge represents the application of a single primitive of an individual vehicle. Allowing the vehicles to move one at a time and simply following the movement order found in the path therefore constitutes a valid schedule. □

Note that the schedule found by the planner is not parallelized, as the calculation of an efficient schedule is delegated to a subsequent step [19]. To gain an intuition of how the number of robots affects the run time of the algorithm, we can reason about how the state space is built and explored. The branching factor grows linearly with the number of robots, as, during each expansion, the planner generates a number of new states equal to the number of robots multiplied by the number of applicable primitives for each robot (Section IV-A). Allowing for concurrent actions could entail that the branching factor could grow exponentially with the number of robots. We leave a detailed complexity analysis of our planner and the exploration of more sophisticated state expansion strategies for future work.

V. Speeding up the search

We designed two methodologies for speeding up the search, *Fast Successor Generation* (FSG) and *Expansion Pruning* (EP). FSG has been designed to decrease the computational costs for collision detection while calculating the set of applicable collision-free primitives, and it can be beneficial regardless of the number of vehicles in the planning problem. EP, on the other hand, is applicable only when the state space encompasses more than one vehicle, and it is used to identify situations in which we can safely avoid expansions. Both strategies do not prune valid solutions.

A. Fast Successor Generation

Given a vehicle of model m, when generating successors for its configuration $c = \langle x, y, \theta, \phi \rangle$, we need to check for collisions all cells swept by every $p \in P_{m,\theta,\phi}$, that is, the set $s_{m,x,y}(p)$ of cells traversed by the footprint of the vehicle when it executes primitive p.[3] However, this procedure might result in cells being checked multiple times in a single expansion. Consider the simplified example in Fig. 1(a), where $P_{m,\theta,\phi} = \{a, b, c\}$, and the vehicle's footprint only sweeps the cells that are swept by each primitive, as shown in the figure. In this case, cell `A1` will be checked three times, once for every primitive that sweeps it. Let $A_{m,\theta,\phi} = \bigcup_{p \in P_{m,\theta,\phi}} s_m(p)$ be the set of cells swept by at least one primitive $p \in P_{m,\theta,\phi}$. FSG guarantees that each cell is checked at most once, reducing the maximum number of checks from $\sum_{p \in P_{m,\theta,\phi}} |s_m(p)|$ to $|A_{m,\theta,\phi}|$ at each expansion. This is because, instead of verifying that each cell in $s_m(p)$ is unblocked for every p, FSG checks every cell in $A_{m,\theta,\phi}$ just once and, if obstructed, marks the primitives that sweep that cell. All primitives left unmarked are collision free.

To do this efficiently, FSG builds offline a directed acyclic graph $G_{m,\theta,\phi} = \langle V_{m,\theta,\phi}, E_{m,\theta,\phi} \rangle$ for each $P_{m,\theta,\phi}$, such that each vertex $v \in V_{m,\theta,\phi}$ contains a set of cells $A_v \subseteq A_{m,\theta,\phi}$ and is uniquely labeled by a set of primitives $P_v \subseteq P_{m,\theta,\phi}$.

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

[3] $s_{m,x,y}(p)$ is calculated by retrieving the set $s_m(p)$, computed offline, and offsetting every cell in it by (x, y) – see Section III-A.

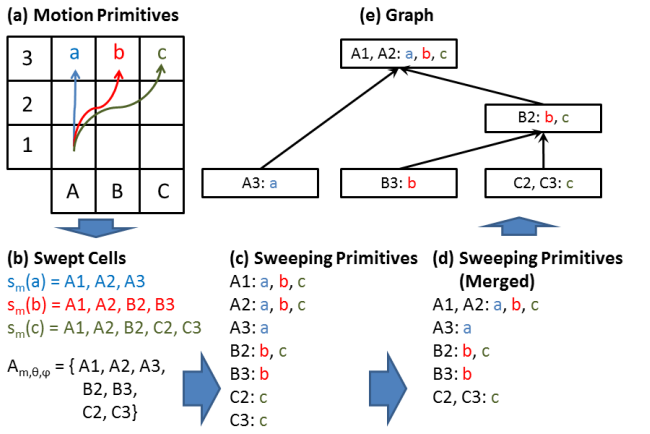

Fig. 1. Generation of graph $G_{m,\theta,\phi}$. Here we assume that the footprint of the vehicle only sweeps the cells that are swept by each primitive. A larger footprint would result in larger sets of swept cells for each primitive.

We construct $G_{m,\theta,\phi}$ as follows (Fig. 1): (1) For each cell in $A_{m,\theta,\phi}$, we calculate the set of primitives that sweep it. (2) We group together the cells that have the same set of sweeping primitives. (3) For each group of cells, we create a vertex v in $G_{m,\theta,\phi}$, such that A_v contains exactly the cells in the group and P_v the primitives sweeping the cells in A_v. (4) For any primitive $p \in P_{m,\theta,\phi}$, a vertex v is created with $P_v = \{p\}$ and $A_v = \emptyset$, unless there already exists a vertex u with $P_u = \{p\}$. (5) We add a directed edge (v, u) to $E_{m,\theta,\phi}$ from vertex v to vertex u iff $P_v \subset P_u$ and there does not exist a vertex w such that $P_v \subset P_w \subset P_u$. By construction, each cell in $A_{m,\theta,\phi}$ appears only in one vertex of $G_{m,\theta,\phi}$, whose maximum depth and maximum number of vertices are, respectively, $|P_{m,\theta,\phi}|$ and $|A_{m,\theta,\phi}|$. Given a vertex $v \in V_{m,\theta,\phi}$, if a cell in A_v is blocked, none of the primitives in P_v is applicable. $G_{m,\theta,\phi}$ has the following properties:

- For any $v, u \in V_{m,\theta,\phi}$, $v \neq u \Rightarrow A_v \cap A_u = \emptyset$.
- For any edge $(v, u) \in E_{m,\theta,\phi}$, $P_v \subset P_u$.
- For any $v, u \in V_{m,\theta,\phi}$, if u is reachable from v, then $P_v \subset P_u$.
- For any $p \in P_{m,\theta,\phi}$, there exists a $v \in V_{m,\theta,\phi}$, such that $P_v = \{p\}$ and the union of A_u for all vertices u reachable from v (including v itself) is equal to $s_m(p)$.

Given a model m and a configuration $c = \langle x, y, \theta, \phi \rangle$, FSG uses $G_{m,\theta,\phi}$ to compute the set of applicable collision-free primitives as follows: (1) All the vertices $v \in V_{m,\theta,\phi}$ are marked as unblocked. (2) Each vertex $v \in V_{m,\theta,\phi}$ is processed in decreasing order of $|P_v|$; v is marked as blocked iff $\exists (v, u) \in E_{m,\theta,\phi}$, where u is marked as blocked, or $\exists (x', y') \in A_v$, where $(x + x', y + y')$ corresponds to an obstacle on the map (offset by (x, y) as the vehicle starts its motion in $c = \langle x, y, \theta, \phi \rangle$). (3) For each unblocked vertex v with $|P_v| = 1$ the corresponding primitive is collision free.

Intuitively speaking, marking a vertex v as blocked means that none of the primitives $p \in P_v$ is collision-free. This is ascertained by first checking if there exists any other vertex u such that $(v, u) \in E_{m,\theta,\phi}$ and u has been marked as blocked, in which case v should also be marked as blocked, as $P_v \subset P_u$. Next, if v has not been marked as blocked, the algorithm checks the cells in A_v. If any of them contains an obstacle,

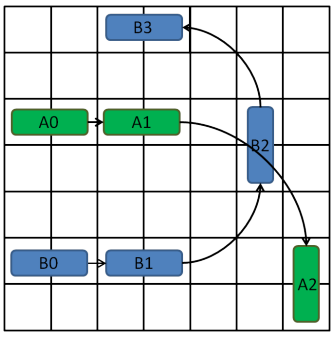

Fig. 2. Example of a multi-robot motion planning problem, where $C_s = \{A0, B0\}$ and $C_g = \{A2, B3\}$. The solution is $\pi_{glob} = (\pi_A, \pi_B)$, where $\pi_A = (p_0^A, p_1^A)$ and $\pi_B = (p_0^B, p_1^B, p_2^B)$.

v is marked as blocked because all of the primitives $p \in P_v$ sweep that cell. The order in which the vertices are processed guarantees that, when v is processed, any vertex u with $P_v \subset P_u$ has already been processed. When a vertex v is marked as blocked, this is propagated down to any vertex u with $P_u \subset P_v$. Hence, if a vertex v with $P_v = \{p\}$ is marked as unblocked when the procedure terminates, p is collision-free.

B. Expansion Pruning

EP is a technique to reduce the number of successors generated during each expansion while exploring the state space in a multi-robot motion planning problem. We have defined the solution to such a problem as a set of sequences of primitives $\pi_{glob} = (\pi_1, \ldots, \pi_N)$, where each π_i represents the individual plan of vehicle i and is of the form $(p_0^i, \ldots, p_n^i)$, where p_j^i is the j-th primitive for vehicle i. As a successor of a state in our state space is the result of the application of a single primitive of one vehicle, starting from C_s there are different but equivalent ways to reach C_g, according to the order in which the vehicles move.

Let us consider the example in Fig. 2. Here, the start state is $C_s = \{A0, B0\}$, the goal state $C_g = \{A2, B3\}$ and the two individual plans $\pi_A = (p_0^A, p_1^A)$ and $\pi_B = (p_0^B, p_1^B, p_2^B)$ constitute a valid solution. However, in the state space, there are different yet equivalent paths which lead from C_s to C_g and the extracted π_A and π_B are equivalent. As a path in the state space is a sequence of single-vehicle moves, two valid paths to reach the solution are $(p_0^A, p_1^A, p_0^B, p_1^B, p_2^B)$ or $(p_0^A, p_0^B, p_1^A, p_1^B, p_2^B)$. Note that $(p_0^B, p_0^A, p_1^B, p_1^A, p_2^B)$ is not a valid path in the state space, as after motion p_1^B, vehicle B is in configuration $B2$, and hence an obstacle for motion p_1^A. In all these paths in the state space, the resulting individual plans π_A and π_B are identical, as is the cost of the global solution. Our joint state space presents a large number of symmetries which slow down the search process, as our algorithm might try to generate a state multiple times. EP breaks some of these symmetries and reduces the number of successor states generated at each expansion.

Definition 6. *A* path *in the state space is a sequence of primitives* $\gamma = (p^0, \ldots, p^K)$ *where each* p^i *is executed by a single vehicle* $\mathrm{v}(p^i)$. *A path* γ *is* valid *iff there exists in the state space a sequence of states* $(C_0, \ldots, C_{K+1})$ *such that each primitive* p_i *is applicable in state* C_i *and connects* C_i *with state* C_{i+1}. *Two paths in the state space are* equivalent

iff, for each vehicle, the sub-sequences of the primitives of this vehicle are identical for both paths.

Definition 7. *Given a bijective function* $pri : \mathtt{v} \rightarrow \mathbb{N}$ *which assigns a unique priority value to each vehicle* $\mathtt{v}$*, two consecutive primitives* p^i *and* p^{i+1} *in a valid path* $\gamma = (p^0, \ldots, p^K)$ *are* swappable *iff* $pri(\mathtt{v}(p^{i+1})) > pri(\mathtt{v}(p^i))$ *and* $\gamma' = (p^0, \ldots, p^{i-1}, p^{i+1}, p^i, p^{i+2}, \ldots, p^K)$ *is valid.*

Definition 8. *A valid path* $\gamma = (p^0, \ldots, p^K)$ *in the state space is* sorted*, iff for all* i *with* $1 \leq i \leq K-1$*,* p^i *and* p^{i+1} *are not swappable,* unsorted *otherwise.*

Theorem 1. *For any valid unsorted path in the state space, there is an equivalent valid sorted path in the state space.*

Proof. We prove this theorem by construction. Given a valid unsorted path in the state space γ, we can swap any swappable pair of primitives, until no more swaps are possible. Due to the priority requirements for a swap, there cannot be any cycles in the swapping process. Therefore, it is guaranteed to terminate. A swap can never change the relative order of moves performed by a single vehicle, as it requires $pri(\mathtt{v}(p^{i+1})) > pri(\mathtt{v}(p^i))$ nor add or remove primitive instances from γ. Therefore, a swap always generates an equivalent valid path in the state space. □

Theorem 1 guarantees that we can prune any valid path in the state space that contains a swappable pair of primitives, as the equivalent valid sorted path will not be pruned.

EP works as follows. For each state C in the state space, with the exception of C_s, we record the primitive p that was applied to generate it. When expanding C, we avoid to generate successors by applying any p' which could create a swappable pair with p. Checking whether p and p' are swappable can be costly, as it requires two potentially costly checks: Whether the configuration of $\mathtt{v}(p)$ before executing p blocks $\mathtt{v}(p')$ from executing p', and whether the configuration of $\mathtt{v}(p')$ after executing p' blocks $\mathtt{v}(p)$ from executing p. Because of the computational overhead that accurate checks would require, EP quickly identifies situations where we can safely avoid generating successors for a vehicle altogether. More specifically, suppose state C is being expanded, which resulted from the application of primitive p executed by vehicle $\mathtt{v}(p)$. We check for each vehicle $\mathtt{v}_i$ if $pri(\mathtt{v}_i) \leq pri(\mathtt{v}(p))$. If this is the case, any application of a primitive p' where $\mathtt{v}(p') = \mathtt{v}_i$ moves is, by definition, not swappable with p. Otherwise, we check the distances between $\mathtt{v}_i$ and the location of $\mathtt{v}(p)$ both before and after the execution of p. If both distances are above the mutual interference range of the vehicles, that is, if they are larger than the sum of the maximum distances that each vehicle could traverse by applying one primitive, p could be swapped with any p' such that $\mathtt{v}(p') = \mathtt{v}_i$ and we can avoid considering vehicle $\mathtt{v}_i$ in the expansion of state C.

VI. Experimental Evaluation

We evaluated our planner in four experimental setups. The first three, *Free Space*, *Crossing* and *Independent Rooms*, were run in simulation and were designed to test the planner's capabilities in specific settings. The fourth scenario was part of a broader evaluation of a complete AGV management system with real robots. In all experiments, we used three models, which represent two different car-like vehicles: SnowWhite (Fig. 6(a)), a lab version of an industrial AGV, and a Linde CiTi Truck (Fig. 6(b)), a retro-fitted industrial forklift. The set of primitives for each model was calculated and automatically reduced in size as described in Section III.

- *SW1*: model based on SnowWhite. $|\Theta_{SW1}| = 8$, $|\Phi_{SW1}| = 1$, $|P_{SW1}| = 224$, grid r_{SW1}: 0.2 meters.
- *SW3*: model based on SnowWhite. $|\Theta_{SW3}| = 8$, $|\Phi_{SW3}| = 3$, $|P_{SW3}| = 736$, grid r_{SW3}: 0.2 meters.
- *CT1*: model based on Linde CiTi Truck. $|\Theta_{CT1}| = 16$, $|\Phi_{CT1}| = 1$, $|P_{CT1}| = 2136$, grid r_{CT1}: 0.2 meters.

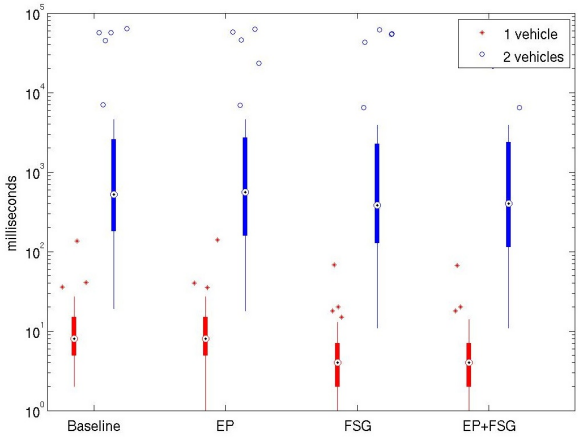

Fig. 3. Free Space scenario: Computation times to find optimal solutions when one or two vehicles are deployed. Each bar aggregates the results of 50 test runs. Note that the y axis is in logarithmic scale.

A. Free Space

We tested our planner in a simulated 20 by 20 meters obstacle-free space, using *SW1*. We performed two sets of 50 test runs each. In each set we deployed, respectively, one and two vehicles in the environment, with randomly chosen start and goal configurations. We repeated each run four times: In the first run, we did not use any method to speed up the search, in the second and third run, we used EP and FSG, respectively, and in the fourth run, we used EP and FSG. The search algorithm employed was A^*, to find optimal solutions. The aggregated execution times are plotted in Fig. 3, where the central mark of each bar is the median, the edges are the 25th and 75th percentiles, the whiskers extend to the most extreme data points not considered outliers and outliers are plotted individually. Most of the single-vehicle instances were solved in less than 10 ms, while when two vehicles were present at the same time, the median time rose to between 100 and 1000 ms. Here, while FSG speeds up the computation, EP does not contribute to lowering the execution times. This is so because the two vehicles often move close to each other and the simple checks performed by EP are not sufficient to prune the number of expansions, while requiring extra time to be calculated.

B. Crossing

In this scenario two vehicles maneuver to pass a crossing. The simulated environment has a size of 10 by 10 meters

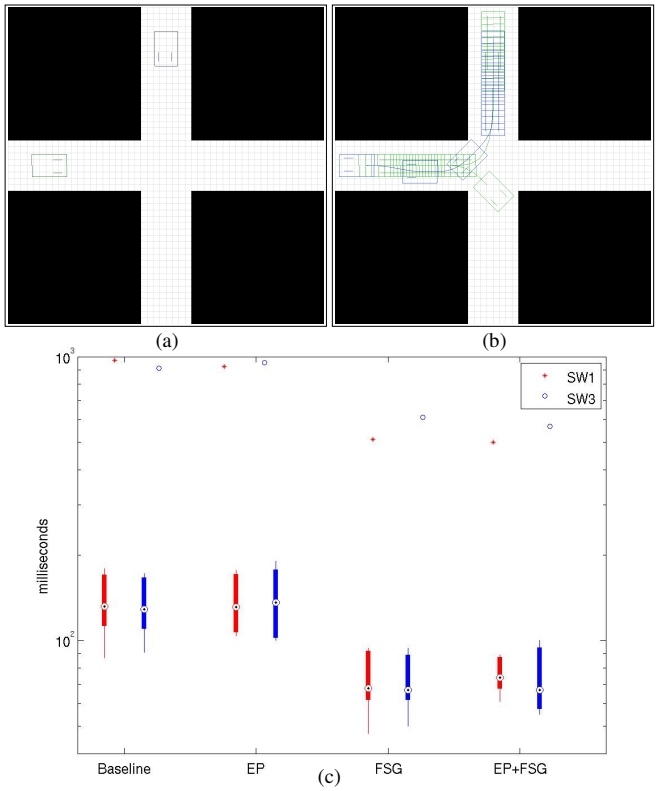

Fig. 4. Crossing scenario. 4(a): Starting configurations in one of the test runs. 4(b): Schedulable joint plan for the two vehicles. 4(c): Aggregate results for the computation times.

and the vehicles' start and goal configurations are placed at one of the ends of the corridors. We performed 11 test runs, without repeating symmetric start/goal combinations. Fig. 4(a) shows the start configurations of one of the runs, where the vehicles are required to swap places: The solution found by the planner ensures that one of the vehicles moves aside to let the second one pass (Fig. 4(b)). We repeated the same test runs with two different models, *SW1* and *SW3*, and we explored the state space using A^*. The resulting execution times are presented in Fig. 4(c). They are consistent with the previous scenario: FSG speeds up the computation (the y axis in Fig. 4(c) is in logarithmic scale), while EP is ineffective when the vehicles move in close quarters. The calculation of the joint plans required less than a second, even when we used model *SW3* with a $|P_{SW3}| = 736$.

C. Independent Rooms

This scenario is intended to test the performance of the planner when the motions of individual vehicles which are far apart are calculated jointly. The test scenario is a 40 by 40 meter area divided into four separated rooms. We tested the planner with an increasing number of vehicles, from 1 to 4, using both *SW1* and *SW3*. For each combination of vehicle model and number of vehicles, we performed 50 test runs, where the start and goal configurations of each vehicle were randomly chosen in separate rooms and, therefore, their motions would most likely not overlap. We used A^* as search algorithm, setting a timeout of 180 seconds per run because the size of the state space could lead to lengthy computations. We then measured the percentage of instances

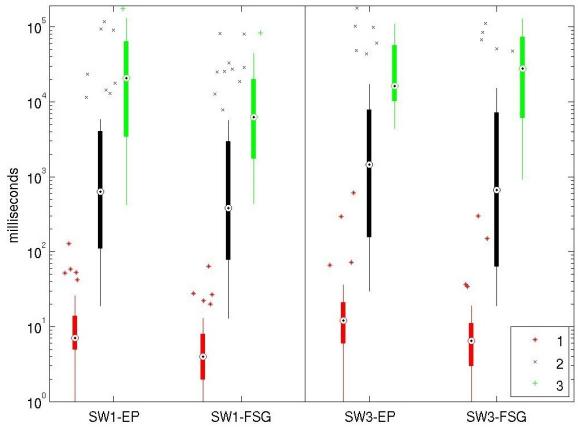

Fig. 5. Independent Rooms: Aggregated results for the computation times.

Vehicles	Baseline	EP	FSG	EP+FSG
1	100%	100%	100%	100%
2	92%	**96%**	92%	**96%**
3	32%	**40%**	32%	**40%**
4	4%	2%	**6%**	2%

TABLE I

INDEPENDENT ROOMS: INSTANCES SOLVED FOR MODEL *SW1*.

Vehicles	Baseline	EP	FSG	EP+FSG
1	100%	100%	100%	100%
2	74%	**80%**	74%	**80%**
3	20%	**30%**	20%	**30%**
4	0%	0%	0%	0%

TABLE II

INDEPENDENT ROOMS: INSTANCES SOLVED FOR MODEL *SW3*.

in which the algorithm provided a solution. As can be seen from the results in Tables I and II (for models *SW1* and *SW3*, respectively), EP produced the best results in terms of solved instances, as its simple expansion pruning technique could be effectively applied. The results also reflect the fact that the state space increases with the size of $|P_m|$. Fig. 5 presents the aggregated computation times over solved instances when the planner is used with FSG and EP.

D. Multi-Robot Motion Planning on a Deployed System

Our final scenario is part of an extensive evaluation of a complete AGV management system, where the planner was used in conjunction with a coordinator [19], [25]. The evaluation was performed in an area comprising several rooms and corridors (Fig. 6(b)). The first part of the evaluation consisted of the continuous coordination of the two vehicles to reach arbitrarily chosen goal configurations. The individual motions were calculated in less than 200 ms by the planner using model *CT1*, then scheduled and executed. The overall system proved to be robust to delays, as the scheduler could dynamically adjust the temporal profiles of the trajectories. A more challenging part of the evaluation consisted of two vehicles switching places. Here, the planner was invoked to calculate the necessary joint maneuvers, and it took less than one second to generate the two motions shown in continuous red lines in Fig. 6(c). The coordinator then calculated a valid temporal profile for them, in which the two vehicles move in parallel to reach their final destinations. This can be seen in the attached video and in Fig. 6(d),

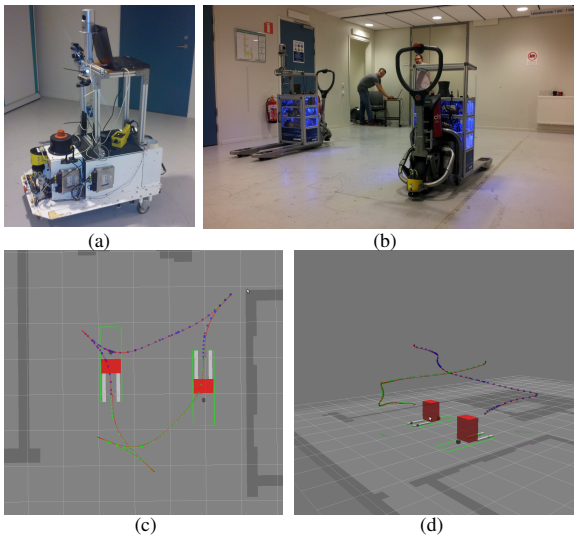

Fig. 6. Deployed system. 6(a),(b): SnowWhite, a scaled lab version of an industrial AGV, and a Linde CiTi Truck, a retro-fitted autonomous industrial forklift. 6(c),(d): the planner calculated a joint plan for two vehicles, so as to allow them to switch places.

where the z axis encodes the time at which each vehicle is scheduled to be at specific points.

VII. Conclusions and Future Work

We presented a new framework which extends lattice-based motion planning for non-holonomic vehicles to multi-robot systems. Our framework generates kinematically feasible joint motions for multiple vehicles. The motions are guaranteed to be schedulable to become collision-free trajectories. We presented two novel techniques which speed up the calculation by decreasing the computational cost of collision checking and by reducing the number of unnecessary state expansions. We have tested our approach in simulation and described a test case with real robots, where our planner is integrated into a full AGV management system. Our work opens up many avenues for future research. First, we intend to provide a full complexity analysis of our approach and to compare it with other approaches. We will investigate different techniques for exploring the state space, inspired by existing algorithms for multi-agent path planning, e.g., M^* [15]. We also intend to further refine our Expansion Pruning technique, so as to increase its effectiveness. Finally, we intend to generalize our approach to more sophisticated representations of the environment.

Acknowledgments

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

References

[1] J. Marshall, T. Barfoot, and J. Larsson, "Autonomous underground tramming for center-articulated vehicles," *Journal of Field Robotics*, vol. 25, no. 6-7, pp. 400–421, 2008.

[2] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[3] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, "Stanley: The robot that won the DARPA grand challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

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

[5] F. Pecora, M. Cirillo, and D. Dimitrov, "On mission-dependent coordination of multiple vehicles under spatial and temporal constraints," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

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

[7] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. on Robotics and Autom.*, vol. 12, no. 4, pp. 566–580, 1996.

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

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

[10] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, B. R. Donald, K. M. Lynch, and D. Rus, Eds. Wellesley, MA: A K Peters, 2001, pp. 293–308.

[11] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The Int. Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[12] E. Szadeczky-Kardoss and B. Kiss, "Extension of the rapidly exploring random tree algorithm with key configurations for nonholonomic motion planning," in *Proc. of the IEEE Int. Conf. on Mechatronics (ICM)*, 2006.

[13] 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.

[14] S. Koenig and M. Likhachev, "D* lite," in *Proc. of the National Conf. on Artificial Antelligence (AAAI)*, 2002.

[15] 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.

[16] 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.

[17] 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.

[18] 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 Autom. (ICRA)*, 2011.

[19] M. Cirillo, T. Uras, S. Koenig, H. Andreasson, and F. Pecora, "Integrated motion planning and coordination for industrial vehicles," in *Proc. of the 24th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2014.

[20] J. P. Desai, J. P. Ostrowski, and V. Kumar, "Modeling and control of formations of nonholonomic mobile robots," *IEEE Transactions on Robotics and Autom.*, vol. 17, no. 6, pp. 905–908, 2001.

[21] A. Kushleyev and M. Likhachev, "Time-bounded lattice for efficient planning in dynamic environments," in *Proc. of the IEEE Int. Conf. on Robotics and Autom. (ICRA)*, 2009.

[22] 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.

[23] M. Pivtoraiko and A. Kelly, "Kinodynamic motion planning with state lattice motion primitives," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[24] 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.

[25] F. Pecora and M. Cirillo, "A constraint-based approach for multiple non-holonomic vehicle coordination in industrial scenarios," in *Proc. of the ICAPS 2012 Workshop on Combining Task and Motion Planning for Real-World Applications (TAMPRA)*, 2012.