2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

Fast, Continuous State Path Smoothing to Improve Navigation Accuracy

Henrik Andreasson[*], Jari Saarinen[*], Marcello Cirillo[*‡], Todor Stoyanov[*] and Achim J. Lilienthal[*]

Abstract— **Autonomous navigation in real-world industrial environments is a challenging task in many respects. One of the key open challenges is fast planning and execution of trajectories to reach arbitrary target positions and orientations with high accuracy and precision, while taking into account non-holonomic vehicle constraints. In recent years, lattice-based motion planners have been successfully used to generate kinematically and kinodynamically feasible motions for non-holonomic vehicles. However, the discretized nature of these algorithms induces discontinuities in both state and control space of the obtained trajectories, resulting in a mismatch between the achieved and the target end pose of the vehicle. As endpose accuracy is critical for the successful loading and unloading of cargo in typical industrial applications, automatically planned paths have not be widely adopted in commercial AGV systems. The main contribution of this paper addresses this shortcoming by introducing a path smoothing approach, which builds on the output of a lattice-based motion planner to generate smooth drivable trajectories for non-holonomic industrial vehicles. In real world tests presented in this paper we demonstrate that the proposed approach is fast enough for online use (it computes trajectories faster than they can be driven) and highly accurate. In 100 repetitions we achieve mean end-point pose errors below 0.01 meters in translation and 0.002 radians in orientation. Even the maximum errors are very small: only 0.02 meters in translation and 0.008 radians in orientation.**

I. Introduction

Automatically Guided Vehicles (AGVs) have been deployed in large numbers for industrial intra-logistic tasks, transporting payload from loading to drop-off locations. Being able to *arrive at a defined pose with high accuracy and precision* is a fundamental requirement for AGV systems. It is especially important since most AGV platforms, e.g., forklift trucks [1] or waist-actuated wheel loaders [2], are non-holonomic. Thus, even in cases when an error in the final pose can be reliably detected, it is not possible for the vehicle controller to correct it over a short distance. According to the AGV system provider Kollmorgen, the required end pose accuracy for picking up pallets is 0.03 m in position and 1 degree (0.017 radians) in orientation, for example.

In current commercial AGV solutions, all paths need to be defined manually before operation. This is a time-consuming, inflexible and costly procedure, which also does not allow for on-line changes during operation to respond to obstacles. In order to enable new applications, it is key to replace this inflexible off-line process with on-line motion planning. State-of-the-art motion planners (based either on Rapidly Exploring Random Trees (RRT), Probabilistic Roadmaps (PRM) or state space lattice search) have so far not been demonstrated to generate trajectories that enable AGVs to achieve the required goal pose accuracy for industrial applications. In addition, the trajectories obtained by current planners are not guaranteed to be directly drivable — discontinuities in vehicle controls and within the trajectories are often left to be handled by the controller. This is a well known problem which has recently received attention [3], [4], but, to-date, no solution has been proposed, which can both guarantee the required accuracy and be used on-line.

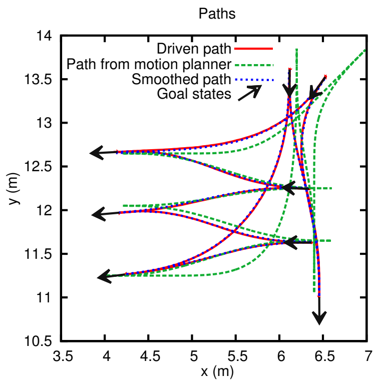

Fig. 1. Example of the problem addressed in this paper. A non-holonomic vehicle has to sequentially drive to a number of given goal poses (black arrows). The figure shows the paths obtained from a lattice-based motion planner (note that these paths contains additional backward/forward motions and the goal is not always where the direction changes), the corresponding smoothed paths and the actual trajectories driven by the vehicle.

The focus of this paper is on on-line motion planning for car-like vehicles given arbitrary end poses. We propose and evaluate a path smoothing approach to improve end pose accuracy obtained with paths generated by a sampling-based motion planner. Our approach also guarantees that the smoothed paths remain collision-free. Achieving end pose accuracy depends on all navigation subsystems. Therefore, we developed a complete navigation system composed of a lattice-based motion planner, the path smoother proposed in this paper, a trajectory generator and a model predictive controller. We compare our system against a state-of-the-art commercial AGV solution which uses manually defined paths. For the comparisons in this paper, we rely on a commercial reflector-based localization module.

The contribution of this paper is two-fold. First, we propose a new path smoothing approach that can be used in a complete, on-line navigation system to produce highly accurate, collision-free motions for car-like vehicles. Our approach does not require a transformation of the input path to a new representation, which is smooth by definition (e.g., splines). Thus, it can directly incorporate state-space constraints in the smoothing process, avoiding the need to

[*] AASS Research Center; Örebro University; Studentgatan 1, 70182 Örebro, Sweden.
[‡] SCANIA AB; Granparksvägen 10, 151 87 Södertälje, Sweden.

verify the path for collisions again after smoothing. In our experiments, we show that our navigation system generates smooth drivable trajectories for arbitrary goal poses in a few seconds, a fraction of the runtime reported by previous approaches [3]. Second, we describe our complete navigation system and present an extensive experimental evaluation. We show the importance of the path smoother and demonstrate that our system can generate and execute smooth paths (see example in Figure 1) with end pose errors comparable to a commercial AGV system based on manually defined paths.

II. Related Work

The industry standard for autonomous navigation is to use predefined trajectories that the AGVs follow strictly. The trajectories are either manually defined or learned through teaching-by-demonstration from a human operator [5], [6]. Although conceptually simple, this approach has drawbacks such as high deployment costs and lack of flexibility. A change in the configuration of a warehouse or an additional loading point, for instance, require the manual definition of a new set of trajectories. Also, if an AGV encounters an unforeseen obstacle during operation, it can only employ very simple strategies (typically stopping until the obstacle moves or is removed). To overcome these drawbacks, many different techniques for automatic path and trajectory generation have been proposed in the past decades.

Combinatorial methods are not very well suited in the presence of differential constraints (e.g., kinematic constraints for non-holonomic vehicles) and analytical solutions cannot effectively cope with obstacles [7]. To overcome these problems, sampling-based approaches have been introduced and studied in recent years. In particular, three families of methods are currently widely used: Probabilistic Roadmaps (PRMs) [8], Rapidly-exploring Random Trees (RRTs) [9], [10], [11] and lattice-based motion planners [12], [13], [14]. All sampling-based approaches have been shown to be effective in high-dimensional configuration spaces. Lattice-based motion planners, in particular, combine the strengths of the approaches discussed above with well studied classical AI graph exploration algorithms, such as A^*, ARA^* and D^*Lite [15]. Differential constraints are incorporated in the search space by means of pre-computed motion primitives, which sample the state space on a regular lattice. The search space is then explored using efficient graph search techniques. However, all sampling-based approaches to non-holonomic motion planning generate paths and trajectories that typically present discretization errors and discontinuities, within the trajectory itself or between the terminal and goal state. This is a known problem which prevents current solutions to achieve the end pose accuracy required by industrial applications. In recent years, several solutions have been suggested for altering the trajectories from sampling-based motion planners [3], [4], which is discussed further in Section IV-E. So far, the methods suggested are computationally too expensive to be used on-line, as smoothing requires time in the order of hundreds or even thousands of seconds [3]. By contrast, our approach is designed to be

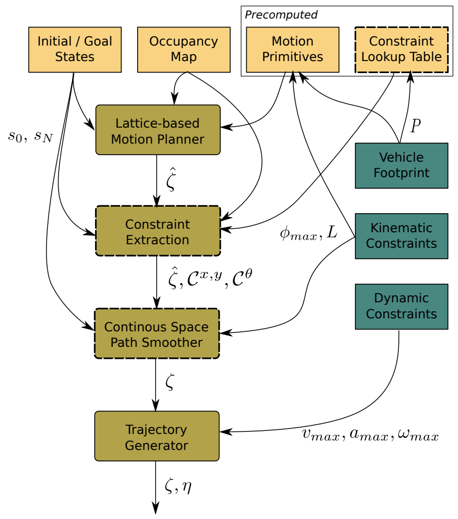

Fig. 2. Overview of the processing steps detailed in Sections III-B, III-C, III-D and III-E and their connections. The novelty in this work lies in the boxes with dashed strokes. s_0 and s_N denote start and goal poses of a vehicle, P the footprint of the vehicle (polygon), $\phi_{\max}$ its maximum allowed steering angle and L the distance between its front and rear axles. $\{v_{\max}, a_{\max}, \omega_{\max}\}$ represent constraints on forward velocity and acceleration, and on steering angle speed. ζ and η are sets of points in state and control space, respectively. Sets of position and angular inequality constraints, denoted $\mathbb{C}^{x,y}, \mathbb{C}^{\theta}$, are extracted using precomputed lookup tables using the vehicle footprint. Motion primitives of a vehicle are generated off-line taking into account its kinematic constraints.

used on-line. In the experimental evaluation we demonstrate that the smoothing step takes less than a few seconds (see Table II).

A different approach to obtain smooth paths is to start with a set of waypoints, and then to switch to a representation that inherently guarantees continuous curvatures. Popular algorithms take as input a set of waypoints, obtained either automatically [16], [17] or by manually driving the vehicle on the desired path [5], [6]. Then an optimization procedure is employed, which operates directly on the parameters of the new representation rather than on the waypoints. Commonly used representations are Quintic splines [18], B-splines [17] or clothoids [19], [20]. Fitting the new representation to the waypoints in general entails a change of the original path and therefore does not guarantee collision-free paths. Thus, a post-processing step is necessary to check whether the path in the new representation is still collision-free [21], [16]. Non-holonomic vehicles have additional constraints on the curvature of the motion, i.e. the maximum steering angle of the vehicle. To guarantee drivable paths, also these constraints require an additional time-consuming verification when a different representation is used for path smoothing.

III. Our Approach

In this work, we consider trajectories for AGVs which operate as a fleet for warehouse automation. At the core of the navigation system is a central vehicle coordinator [22], which can directly control the speed of each autonomous vehicle. Thus, the paths and speed profiles of the controlled vehicles have to be computed separately, in order to allow the coordinator to prevent deadlocks. This is a desirable separation whenever multiple AGVs need to be coordinated.

For our system we adopt the four-step approach shown in Figure 2. First, we calculate kinematically drivable paths with a lattice-based motion planner for each vehicle. Next, we extract an envelope of free space around the planned path, represented in the form of convex polyhedral constraints on the vehicle state. Then, the path and the corresponding constraints are processed by the path smoother proposed in this paper, resulting in a collision-free continuous drivable path. Finally, a trajectory generator associates speed profiles consistent with dynamic and coordination constraints to the smoothed paths and generates the final trajectories that vehicle controllers execute [23].

A. Problem Formulation

In this paper we consider car-like vehicles (standard trucks, forklifts, etc.) that are commonly found in indoor production sites. The control space $u = (v, \omega)$ of such vehicles is composed of forward speed v and steering velocity ω. The state space $s = (x, y, \theta, \phi)$ consists of all vehicle configurations composed of 2D position $[x, y]$ of its reference point, heading θ and steering angle ϕ. The state transition of car-like vehicles $\dot{s} = f(s, u)$ is computed as follows:

$$\begin{aligned}
\dot{x}_i &= v_i \cos\theta_i, \\
\dot{y}_i &= v_i \sin\theta_i, \\
\dot{\theta}_i &= v_i \tfrac{\tan\phi_i}{L}, \\
\dot{\phi}_i &= \omega_i,
\end{aligned} \tag{1}$$

where L is the distance between front and back axles. Let ζ and η be sets of N discretized points in state space and control space respectively, defined as:

$$\begin{aligned}
\zeta = \{s_i\} &= \{(x_i, y_i, \theta_i, \phi_i)\}, \\
\eta = \{u_i\} &= \{(v_i, \omega_i)\}.
\end{aligned} \tag{2}$$

Given an initial state $s_0 = (x_0, y_0, \theta_0, \phi_0)$ and a goal state $s_N = (x_N, y_N, \theta_N, \phi_N)$ for a vehicle, we want to calculate a set of N control points η and the corresponding state points ζ, which can take the vehicle precisely from s_0 to s_N, without colliding with any known obstacle. We assume a fixed time step $\Delta T = \frac{T}{N}$, where T is the time for reaching the goal. A solution to the problem is valid if also additional vehicle-dependent constraints are respected, such as the maximum allowed steering angle ϕ_{max}, the maximum steering velocity ω_{max}, the maximum forward and reverse velocity v_{max} and the maximum acceleration $|\dot{v}_i| \leq a_{\text{max}}$.[1]

[1]In our experiments, the x_0, y_0 and θ_0 components of the start state s_0 are obtained directly from an off-the-shelf localization system, while the steering angle ϕ_0 is taken from absolute encoder readings. The goal state is always assumed to have a steering angle component $\phi_N = 0$.

B. Lattice-based Motion Planner

The first step of our approach is a lattice-based motion planner [14], which quickly computes kinematically feasible paths, optimized with respect to a cost function that considers distance traveled and penalizes backwards and turning motions. 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. Each vertex of the lattice represents a valid pose of the vehicle $\hat{s} = (x, y, \theta, \phi)$, while each edge encodes a motion which respects the non-holonomic constraints of the vehicle. The lattice is constructed over a grid whose cells have a fixed side length, while θ and ϕ at each lattice vertex are constrained to pre-define finite sets of headings and steering angles, respectively.

The planner uses a set of pre-computed, kinematically feasible motion primitives, which are repeatedly applied to build the lattice. Information about the static obstacles in the environment is provided to the planner by an occupancy map and is used to prune the search graph to obtain collision-free paths. The motion primitives are automatically generated to fully capture the mobility of the vehicle and then reduced for efficiency purposes, as described in [24]. The lattice graph is then explored using ARA^* [25]. Effective heuristic functions [26], as well as pre-computed vehicle footprints for each motion primitive are employed to speed up the exploration of the lattice.

Given a start and a goal state (s_0, s_N), the motion planner generates an obstacle free, kinematically drivable path $\hat{\zeta}$, which means that the steering angle constraint $-\phi_{\text{max}} \leq \phi \leq \phi_{\text{max}}$ (Eq. 8) is respected. Note that the planner at this stage does not generate the controls for the vehicle. The path $\hat{\zeta}$, however, brings the vehicle from a discretized start state $\hat{s}_0$ to a discretized goal state $\hat{s}_N$, where $\hat{s}_0$ and $\hat{s}_N$ represent the closest states on the lattice to s_0 and s_N, respectively. This introduces an error on the order of the lattice discretization. Moreover, $\hat{\zeta}$ is by construction guaranteed to be drivable by the vehicle at a nominal speed and it is C^1-continuous, but not necessarily C^2-continuous — i.e., the rate of change of the steering angle of the vehicle can be discontinuous. Throughout this paper the grid and angular discretization used is 0.2 meters and $\pi/8$ radians respectively.

C. Constraint Extraction

In order to produce provably collision-free smoothed paths, our approach needs to constrain the search space of robot states to collision-free states only. While this is a difficult problem in general, we can utilize the collision-free path $\hat{\zeta}$ provided by the motion planner and associate an envelope of space around it in which path nodes can be safely moved. Starting from each $\hat{s}_i \in \hat{\zeta}$, we define a set of inequality constraints to which the corresponding state point in the smoothed path should adhere. Naturally, "larger" spatial constraints will allow greater flexibility into the smoothing step and more likely a better path.

We use two types of constraints: constraints on the position of the vehicle's reference point $\mathcal{C}^{x,y} = \{C^{x,y}_{ij}\}$ and

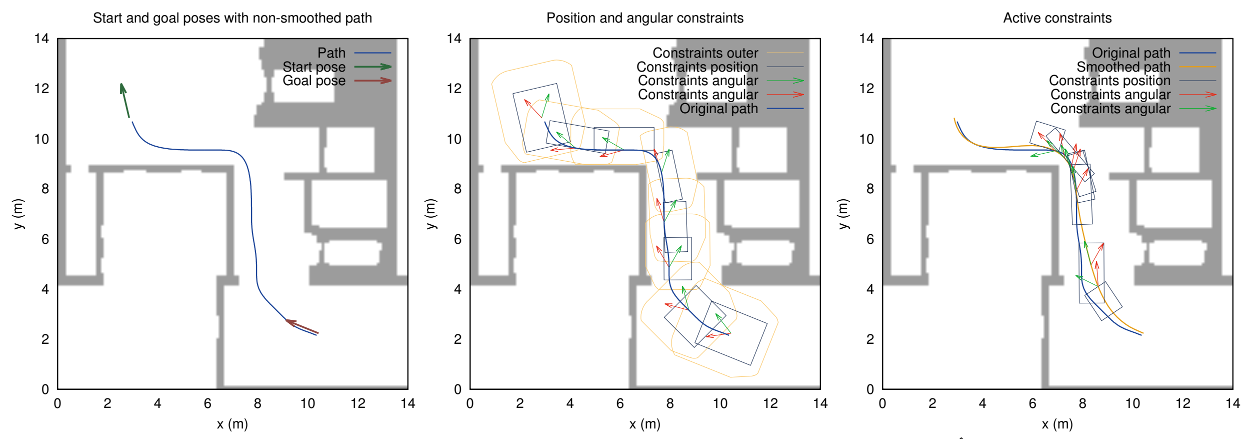

Fig. 3. Sequential steps in path smoothing. Left: Given start and goal poses, the motion planner calculates the path $\hat{\zeta}$. Center: Spatial constraints extraction. Position constraints (blue boxes), angular constraints (red and green arrows) and their spatial combination (yellow line) are calculated for each $(\hat{x}_i, \hat{y}_i) \in \hat{\zeta}$. For the sake of clarity, the figure represents only a sub-set of state points $(\hat{x}_i, \hat{y}_i) \in \hat{\zeta}$. Right: Active constraints plotted on the smoothed path.

constraints on its orientation $\mathcal{C}^\theta = \{C_i^\theta\}$, where $0 \leq i \leq N$ represents the state to which the constraint refers and $0 \leq j \leq M$ indexes a number of spatial constraints referring to the same state $\hat{s}_i$. Orientation constraints limit the orientation the vehicle can assume (as shown in Equation 3), where ${}^{\mathrm{lb}}C_i^\theta$ and ${}^{\mathrm{ub}}C_i^\theta$ denote the lower and upper bounds for the vehicle's orientation in the smoothed state s_i. Position constraints are formulated as detailed in Equation 4, where ${}^{\mathrm{A0}}C_{ij}^{x,y}$, ${}^{\mathrm{A1}}C_{ij}^{x,y}$ and ${}^{\mathrm{b}}C_{ij}^{x,y}$ constitute a linear inequality which bounds the area in which the vehicle's reference point can lie. Furthermore, the set of linear inequalities $C_{ij}^{x,y}$ referring to the same state $\hat{s}_i$ are constructed as to define a convex polygon, thus facilitating the optimization problem described in the next subsection.

$$ {}^{\mathrm{lb}}C_i^\theta \leq \theta_i \leq {}^{\mathrm{ub}}C_i^\theta \tag{3} $$

$$ {}^{\mathrm{A0}}C_{ij}^{x,y} x_i + {}^{\mathrm{A1}}C_{ij}^{x,y} y_i \leq {}^{\mathrm{b}}C_{ij}^{x,y} \tag{4} $$

A constraint set $\mathcal{C}_i = \{\mathcal{C}_i^{x,y}, \mathcal{C}_i^\theta\}$ defines an area a_i which can be obtained by sweeping the footprint of the vehicle over all the positions allowed by $\mathcal{C}_i^{x,y}$ and with all the orientations allowed by $\mathcal{C}_i^\theta$ (see yellow-bounded areas in Figure 3, Center).

The selection of appropriate constraints is not trivial: given the free area around a specific state, assigning larger bounds to one of the two sets of constraints limits the values which can be assigned to the other set. Most importantly, the selection of appropriate constraints must be fast.

To quickly identify an adequate set of constraints when a new path $\hat{\zeta}$ is generated by the planner, we proceed as follows: For each type of vehicle, we *pre-compute* a set of constraints $\mathcal{C} = \{\mathcal{C}_k\}$, where $1 \leq k \leq K$ and each $\mathcal{C}_k \in \mathcal{C}$ is in turn a set of 4 position constraints (representing a rectangle) and an orientation constraint. For each set $\mathcal{C}_k$ we can then also compute the relative sweeping area a_k, and we can store it in a lookup table along with its weight $w_k = g(\mathcal{C}_k^{x,y})h(\mathcal{C}_k^\theta)$, where g returns the enclosed area formed by the position constraints and h the width of the orientation constraints. For each state $\hat{s}_i \in \hat{\zeta}$, we select the collision free constraint set $\mathcal{C}_k$ with the highest weight. Note that using this method we perform a one-time pre-computation of a large number of constraint sets for each type of vehicle. These sets can then be used on-line, avoiding the complex and time consuming collision checks required by other approaches [21], [16].

In all the experiments presented in this paper, the number of pre-computed constraints is set to $|\mathcal{C}| = 28224$. The position constraints were pre-calculated by allowing for different forward/backward and side displacements of the reference point of the vehicle – 7 offsets in forward/backward direction, from 0.1 to 1.2 meters, and 6 offsets sideways, from 0.01 to 0.9 meters. The orientation constraints were computed using 4 lower and upper bounds – $\pi/180$, $\pi/36$, $\pi/12$ and $\pi/6$ radians. To have large constraints in the forward/backward direction is benefitial to reduce the path length when the vehicle changes traversal direction. The sweeping area a_k of each subset $\mathcal{C}_k$ was encoded as a template on a grid with the same resolution of the occupancy map containing the obstacles in the environment. For each state point $\hat{s}_i$, the occupancy map is aligned with the templates in the lookup table to avoid performing the same transformation repeatedly. A constraint set is considered collision free if its template does not contain any occupied cell in the aligned occupancy map.

D. Continuous Space Path Smoother

As mentioned above, the path $\hat{\zeta}$ obtained by the motion planner presents two major problems: First, the start and goal states used by the planner are discretized and do not necessarily correspond to the given initial and end state (see Figure 1); and second, $\hat{\zeta}$ is not necessarily C^2-continuous. While the latter issue can be handled and corrected by the controller, the accuracy of the final vehicle pose with respect to the target pose would be at best on the order of the grid discretization of the planner (as demonstrated in the experiments in Section IV). In order to solve these problems and improve navigation, our approach modifies $\hat{\zeta}$.

We begin by formulating the above problems as constraints on the states of $\hat{\zeta}$. First, the start and end states of the modified path need to correspond precisely to the current vehicle state $\bar{s}_0$ and the actual goal state $\bar{s}_N$:

$$\begin{aligned} \bar{s}_0 - s_0 &= 0, & (5)\\ \bar{s}_N - s_N &= 0 & (6) \end{aligned}$$

These constraints address inaccuracies in the initial and end state that occur due to discretizations in the lattice-based motion planner. The additional constraints in Eqs. 7-11 make sure that the path conforms to the kinematic constraints of the vehicle and the constraints on vehicle controls:

$$\begin{aligned} s_{i+1} = \hat{f}(s_i, u_i),\ i = 1, \ldots, N-1, && (7)\\ \phi_{\text{min}} \leq \phi \leq \phi_{\text{max}}, && (8)\\ -v_{\text{max}} \leq v \leq v_{\text{max}}, && (9)\\ -\omega_{\text{max}} \leq \omega \leq \omega_{\text{max}}, && (10)\\ -a_{\text{max}} \leq a \leq a_{\text{max}}. && (11) \end{aligned}$$

The constraints in Eqs. 5-11 define the set of all possible executable trajectories to bring the vehicle from a given start to a goal state without considering obstacles. An uninformed search through this set of feasible trajectories is a very inefficient way of solving the continuous space motion planning problem in the presence of obstacles. The key to the efficiency that we obtain with the proposed approach is that an uninformed search is not necessary. From the lattice-based motion planner we already have a cost-optimal, obstacle-free path $\hat{\zeta}$. Although this path generally lies outside the feasible set defined by Eqs. 5-11 we can use it to initialize the variables $\{s_i\}_{i=0\ldots N}$, and to compute the spatial and angular constraint sets $\mathcal{C}_{i=0\ldots N}$ using the pose (x_i, y_i, θ_i) from s_i. Here, we only focus the optimization on generating a path and not on obtaining the fastest trajectory. This is done in order to reduce the complexity of the problem. The decoupling of the generation of the control set from the path is further motivated by the fact that the fastest trajectory profile may not necessarily be the optimal one in a multi-robot navigation scenario [23].

The path $\hat{\zeta}$ is assumed to have state points separated by an approximately equal distance (in our experiments we used a separation of 0.2 m) and we set an equidistant time ΔT between consecutive state points to an arbitrary constant positive value. The time T to traverse the full path is thus set to a constant value of $N\Delta T$. The goal is to minimize the total distance traveled and the total amount of turning applied on the steering wheel. We formulate the objective as a function of the control values:

$$\begin{aligned} \underset{\zeta,\eta}{\text{minimize}} \quad & \sum_{i=0}^{N} v_i^2 + \alpha \sum_{i_1}^{N} \omega_i^2 \\ \text{subject to} \quad & \\ & s_{i+1} = \hat{f}(s_i, u_i),\ i = 0, \ldots, N-1 \\ & \bar{s}_0 - s_0 = 0 \\ & \bar{s}_N - s_N = 0 \\ & \phi_{\text{min}} \leq \phi \leq \phi_{\text{max}},\ i = 0, \ldots, N \\ & {}^{\text{lb}}C_i^{\theta} \leq \theta_i \leq {}^{\text{ub}}C_i^{\theta},\ i = 0, \ldots, N \\ & {}^{\text{A0}}C_{ij}^{x,y} x_i + {}^{\text{A1}}C_{ij}^{x,y} y_i \leq {}^{\text{b}}C_{ij}^{x,y}, \\ & i = 0, \ldots, N,\ j = 0, \ldots M \end{aligned}$$

where α is a weighting factor ($\alpha = 1$ was used in our experiments). Since we have a fixed ΔT between state points the above formulation minimizes a combination of total distance traveled and total amount of turning of the steering wheel. The objective will try to keep an equidistant separation between the state points, similar to the initial path $\hat{\zeta}$ which is assumed to be equidistant, and therefore the number of active constraints due to forward or backward movements of the state points along the path will be limited. Ideally, only the side constraints should be activated during the optimization step. Forward and backward bounds are essential to guarantee collision free paths, but should preferably not be active apart from when the vehicle changes directions. As the computation of new constraints covering free-space is quite costly (in the order of approx. 20 ms per meter path length) we only want to perform this step once. In principle though one could recompute the constraints $\mathcal{C}_{1\ldots N-1}$ during the optimization step.

The approach was formulated and solved using the ACADO Toolkit [27], applying a direct multiple shooting method. Please note that the output of this optimization phase is a trajectory (ζ, η). Since the time parameter T was set arbitrary and no additional constraints (Eq. 9-11) were enforced on the control values η, they are not dynamically feasible and needs to be recomputed. Here we rely on a subsequent trajectory generation step.

E. Trajectory Generator

The trajectory generator takes as input the states computed by the path smoother and it works in a similar way as described in [28], [16]. More specifically, it assigns the largest possible velocity to each state in the path within the given constraints on initial and final velocities, steering velocity, speed and acceleration.

The output of this module is a trajectory with a fixed ΔT of 60 ms, which is the input for the controller. In the current implementation we use linear interpolation to compute state points, which is effective since the distance between interpolation states is small.

F. Model Predictive Controller

In our vehicle navigation system, we use a Model Predictive Controller (MPC) [23]. The core idea of this type of

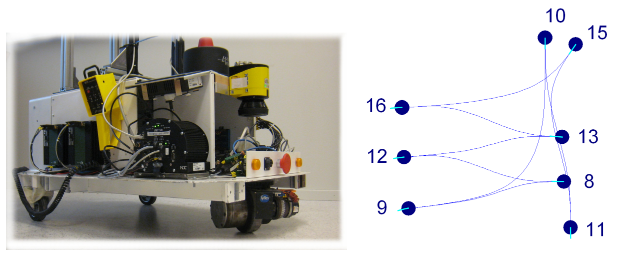

Fig. 4. a) AGV test platform used in the experiments. b) Test layout for the comparison of the systems: the blue dots indicate goal poses identified by an ID. The turquoise line in the dots shows the goal headings.

controllers is to model how the states of the vehicle evolve over a time preview window, given a set of control inputs. The controller then optimizes the control output using an objective function based on the trajectory to follow. In our implementation we use a preview window of 25 control steps and each control step has a duration of 60 ms. The controller gains were set to equally weight heading and distance. This is reflected in the results presented in Figure 5: the heading and distance errors are quantitatively very similar. As the controller is not the main focus of this paper, we use it with default parameters as a "black box" in the experimental evaluation.

IV. Experimental Evaluation

We evaluated our navigation system using a test vehicle with a combined steer and drive wheel configuration (Figure 4) in a small-scale test environment at Örebro University. The vehicle kinematics are the same as those of standard fork lift trucks. A commercial reflector-based localization solution was deployed which guarantees localization accuracy of less than 1 cm in translation and 0.001 radians in orientation. Using the on-board control system, we can access encoder and localization data and we can set steer and drive commands. All of the remaining components of our system run on a standard laptop with an i7-2860QM CPU at 2.50GHz.

The overall goal of our experimental evaluation is to demonstrate the accuracy achieved while following the on-line generated trajectories and the reliability of our results. We structure our evaluation in several consecutive parts. First, we compare the controller of our system with the controller of a commercial solution over identical, manually crafted trajectories. This set of tests is necessary to guarantee that the results obtained are comparable when using different approaches to path generation and are not biased by different performances at the controller level. The following evaluation step is most crucial: once established that the two controllers have comparable capabilities, we extract the goal poses from the pre-defined paths and use them as input to our system. This means that we compare our approach, where the paths are automatically generated, with the performance of the commercial system which needs hand coded paths. These experimental runs allow us to demonstrate that our approach can entirely substitute the commercial system without loss of accuracy. Furthermore, we test the robustness of our system over randomly generated goal poses, evaluate the system over longer distances and with additional obstacles.

A. Comparison with commercial controller

We first created a set of B-spline parametrized trajectories T_{eval} (Figure 4b) with an AGV layout drawing tool provided by Kollmorgen. Special care was taken to make the trajectories as smooth as possible. We then used both controllers to follow the trajectories. Each trajectory was executed 10 times. The AGV controller can follow the given trajectories directly, whereas our controller extracts the paths P_{eval} from the layout trajectories T_{eval} and attaches speed profiles to them, as provided by our trajectory generator (Section III-E). The tracking performance is shown in the first two rows of Table I. In particular, we show that the performance is comparable with respect to the final forward error, side error and heading error.

B. Path smoothing of automatically generated paths

After we have established comparable performance of the two controllers, we now evaluate the path smoothing component.

We extracted the goal poses from the same paths P_{eval} employed in Section IV-A and used them as targets for our lattice-based motion planner. The output of the planner is passed to the path smoother and then executed by the MPC controller. Table I (third and forth rows) shows results obtained when the motion planner is used with and without any path smoothing. Ideally, the automatically generated paths should allow for a comparable pose accuracy as the manually defined paths P_{eval}. This is indeed the case in our tests. Please note that (in row 4), where the paths are not post-processed by the smoother, the required goal state is set as the last point in the trajectory, thus allowing the controller to correct the state error within the preview window. It can be seen that, in this case, the controller can compensate more effectively for errors in the direction of motion.

A key evaluation metric for AGV systems is their ability to repeatedly reach the same goal pose given a specific path. In an industrial scenario with manually predefined paths, precision is often more important than accuracy since a bias in the end pose can be compensated by adding the corresponding offset to the goal pose. For on-line motion planning, however, the key evaluation metric is end pose accuracy. The standard deviation is shown for different goals in Figure 5 including a plot with all reached poses for one goal.

C. Evaluation over randomly chosen goals

We further tested the robustness of our system with arbitrary goal poses and over longer distances, by generating a random set of 50 goal poses in two regions of the test environment, marked with dashed lines in Figure 6. The vehicle traversed back and forth between poses, from one region to the other, for a total of 100 stops. Side, forward and heading errors are shown in the bottom two rows of Table I, while computation times are presented in Table II (mean,

TABLE I

FORWARD AND SIDE TRANSLATION ERRORS AND ORIENTATION ERRORS

Method	forward error mean (m)	forward error std (m)	forward error max (m)	side error mean (m)	side error std (m)	side error max (m)	heading error mean (rad)	heading error std (rad)	heading error max (rad)
predef. path, AGV controller	0.0168	0.0027	0.0224	0.0047	0.0028	0.0098	0.0060	0.0044	0.0138
predef. path, MPC controller	0.0025	0.0018	0.0080	0.0054	0.0044	0.0150	0.0013	0.0010	0.0041
planned path with smoothing	0.0018	0.0011	0.0048	0.0057	0.0040	0.0150	0.0026	0.0018	0.0088
planned path without smoothing	0.0273	0.0299	0.0886	0.0521	0.0211	0.1116	0.0621	0.0602	0.1783
100 random goals short	0.0017	0.0011	0.0047	0.0090	0.0056	0.0236	0.0018	0.0015	0.0067
100 random goals long	0.0016	0.0011	0.0058	0.0087	0.0055	0.0202	0.0018	0.0015	0.0082

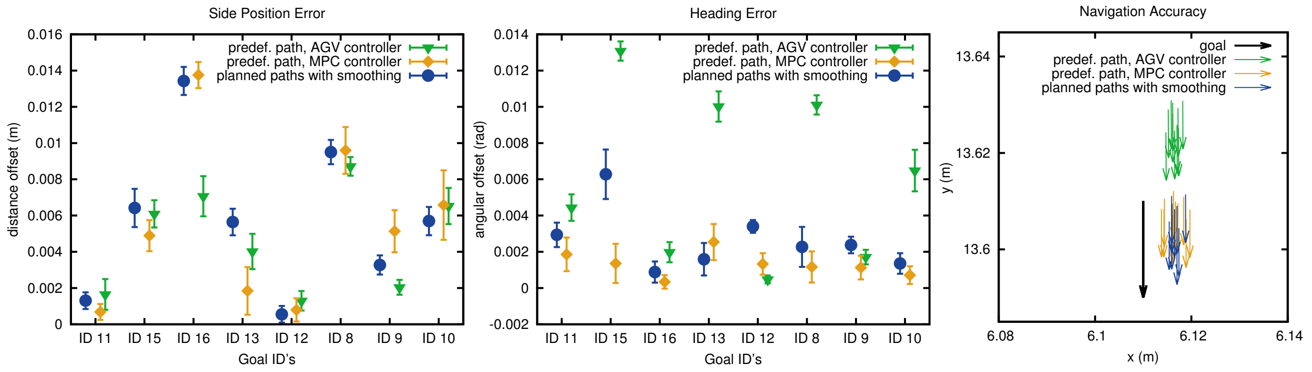

Fig. 5. Navigation pose accuracy. Left: side distance errors. Middle: heading angle errors. The goal IDs refer to Figure 4. Note that the paths generated by the planner and not processed by the smoother present errors one order of magnitude larger than the ones in the figure and therefore are omitted. Right: End pose accuracy evaluated on goal pose ID 10.

TABLE II

COMPUTATIONAL TIME FOR MOTION PLANNING AND PATH SMOOTHING [SHORT/LONG] (SECTION IV-C).

	mean (s)	std (s)	max (s)
Motion planning	0.411/1.995	0.350/1.675	1.581/6.507
Constraint extraction	0.124/0.338	0.029/0.049	0.197/0.465
Path smoothing	0.472/6.643	0.252/3.216	1.416/18.551

TABLE III

PATH STATISTICS [SHORT/LONG] (SECTION IV-C).

	total turning mean (rad)	total turning std (rad)	total distance mean (m)	total distance std (m)
planned path	2.78/5.71	0.78/1.31	6.20/17.08	1.45/2.43
smoothed path	2.03/2.47	0.69/1.00	4.47/16.20	0.84/2.04

max and standard deviation). Table III presents statistics of the length and the total amount of turning in the paths over the different runs. The proposed path smoothing approach results in a large improvements in both aspects and even the max error is below 3cm.

D. Obstacle course

This section shows an example of longer navigation with additional obstacles, see Figure 6. This example gives an impression how the proposed approach behaves in a more challenging scenario. As expected the number of active constraints is higher and the smoothed path deviates less from the original path due to the more limited space compared the run without obstacles.

E. Comparison to Related Work

Both works reported in [3], [4] compute a feasible trajectory by perturbing the original path. This is done by inserting a set of actions in order to reach the goal or by inserting additional segments [3], or by performing stretching operation on segments [4]. Note, however, that none of this operations performs any smoothing along the trajectory. The obtained feasible trajectory furthermore contains a gap offset between the actual goal pose and the planned goal pose (the gap offset acceptance thresholds used in [4] were 0.01 given in a combined meters and radians measure).

To compare the approach with the results presented in [3] we replicated one of their environments, see Figure 6. The environment was scaled down to fit the smaller sized vehicle that was used in [3]. We are especially interested in a comparison of the computational time. It is important to stress that the motion models used in [3] are more complicated to include dynamics, the models used here is the same as in [4]. Unfortunately no path smoothing criterion is mentioned in [3] such as traveled distance and amount of turning applied. The best run time reported in [3] is 3400 seconds running a on 2GHz Linux PC. Compared to this using a discretization size of 0.4 meters the proposed smoothing approach took 31.40 seconds, the path planning time to compute the original path was 17.46 seconds. With a discretization step of 0.2 meters the computation time was 167.83 seconds.

V. CONCLUSIONS AND FUTURE WORK

We have presented a complete navigation system for autonomous, non-holonomic vehicles in industrial settings and we have introduced a novel, fast path smoothing approach, which is applied to the output of a lattice-based motion planner. Lattice-based motion planners, as it is the case with all sampling-based approaches to motion planning,

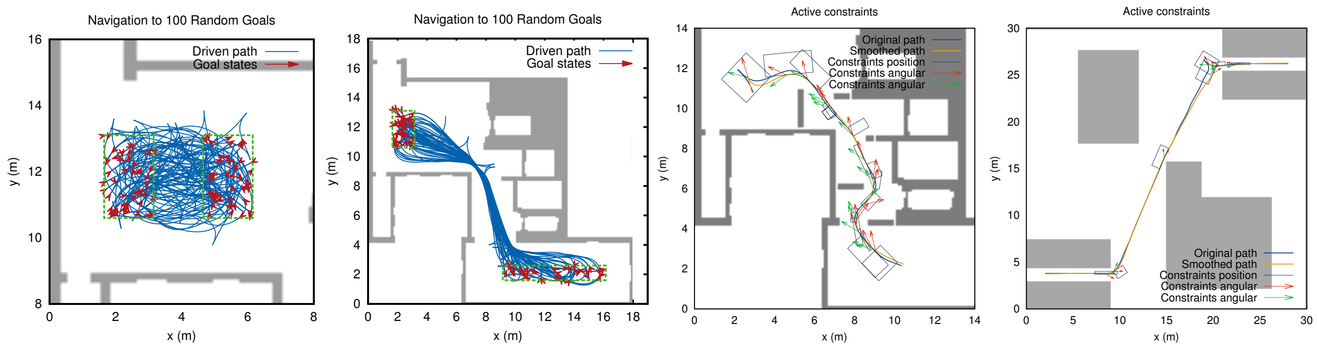

Fig. 6. Left/Center left: Path driven while navigating to 100 random selected goals (short/long). Center right: additional obstacles added into the map using the same start and goal pose as in Fig 3. Right: test environment used in [3].

produce motions that present discontinuities which lead to insufficient accuracy and precision in industrial applications. Our approach has two major advantages: it can work on-line and uses the same state representation as the motion planner. It can thus directly incorporate state-space constraints in the smoothing process and therefore avoid the need to verify the path again after smoothing. This enables direct initialization of the path smoother with the automatically generated obstacle-free paths. Curvature checks on the final path are not necessary and we can directly include additional constraints in the optimization process. This opens up new possibilities to constrain the vehicle when it approaches the goal state. Constraining steering more in the last segments of the path could, for instance, improve pose accuracy even further. This, however, is left for future work. Another interesting avenue for further investigation will be to apply our approach to other planners, RRT- and PRM-based, to ascertain the generalizability of our methodology.

References

[1] A. Bouguerra, H. Andreasson, A. J. Lilienthal, B. Åstrand, and T. Rögnvaldsson, "Malta: A system of multiple autonomous trucks for load transportation," in *Proc. of the European Conf. on Mobile Robots (ECMR)*, 2009.

[2] M. Magnusson and H. Almqvist, "Consistent pile-shape quantification for autonomous wheel loaders," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[3] P. Cheng, E. Frazzoli, and S. LaValle, "Improving the performance of sampling-based motion planning with symmetry-based gap reduction," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 488–494, 2008.

[4] K. M. Seiler, S. P. Singh, S. Sukkarieh, and H. Durrant-Whyte, "Using lie group symmetries for fast corrective motion planning," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 151–166, 2012.

[5] T. Hellstrom and O. Ringdahl, "Follow the past: a path-tracking algorithm for autonomous vehicles," *Int. Journal of Vehicle Autonomous Systems*, vol. 4, no. 2, pp. 216–224, 2006.

[6] 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.

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

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

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

[10] 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.

[11] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "RRT*-smart: Rapid convergence implementation of RRT* towards optimal solution," in *Proc. of the Int. Conf, on Mechatronics and Automation (ICMA)*, 2012.

[12] D. Ferguson and M. Likhachev, "Efficiently using cost maps for planning complex maneuvers," in *Proc. of the ICRA Workshop on Planning with Cost Maps*, 2008.

[13] 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.

[14] M. Cirillo, T. Uras, and S. Koenig, "A lattice-based approach to multi-robot motion planning for non-holonomic vehicles," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

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

[16] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[17] M. Walther, P. Steinhaus, and R. Dillmann, "Using B-splines for mobile robot path representation and motion control," in *Proc. of the European Conf. on Mobile Robots (ECMR)*, 2005.

[18] C. Sprunk, B. Lau, and W. Burgard, "Improved non-linear spline fitting for teaching trajectories to mobile robots," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2012.

[19] D. K. Wilde, "Computing clothoid segments for trajectory generation," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[20] M. Brezak and I. Petrovic, "Path smoothing using clothoids for differential drive mobile robots," in *Proc. of the 18th IFAC World Congress*, 2011.

[21] C. Sprunk, B. Lau, P. Pfaff, and W. Burgard, "Online generation of kinodynamic trajectories for non-circular omnidirectional robots," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[22] 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*, 2014.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[28] V. F. Munoz and A. Ollero, "Smooth trajectory planning method for mobile robots," in *Proc. of the Conf. on Computational Engineering in Systems Applications*, 1996.