Autonomous Robots (2016), Preprint
The final publication is available at Springer via http://dx.doi.org/10.1007/s10514-016-9557-1

An Accurate and Efficient Navigation System for Omnidirectional Robots in Industrial Environments

Christoph Sprunk · Boris Lau · Patrick Pfaff · Wolfram Burgard

Received: 22 January 2015 / Accepted: 15 February 2016

Abstract Enhanced logistics is widely regarded as a key technology to increase flexibility and cost efficiency of today's factories. For example, fully autonomous transport vehicles aim to gradually replace conveyor belts, guided vehicles, and manual labor. In this context, especially omnidirectional robots are appealing thanks to their advanced maneuvering capabilities. In industrial applications, however, accuracy as well as safety and efficiency are key requirements for successful navigation systems. In this paper, we present an accurate navigation system for omnidirectional robots. Our system includes dedicated modules for mapping, localization, trajectory generation and robot control. It has been designed for accurate execution by devising smooth, curvature continuous trajectories, by planning appropriate velocities and by accounting for platform and safety constraints. In this way, it completely utilizes the maneuvering capabilities of omnidirectional robots and optimizes trajectories with respect to time of travel. We present extensive experimental evaluations in simulation and in changing real-world environments to demonstrate the robustness and accuracy of our system.

Fig. 1 The KUKA Moiros robot using our system for accurate omnidirectional navigation during a demonstration at the Hannover Messe 2013 fair. Our system accounts for platform and safety constraints and reliably navigated for several days in this inspection task for wind turbine blades.

1 Introduction

Automated transportation is a cornerstone functionality for logistics in today's highly automated factories. Starting from conveyor belts, industry is gradually moving to automated ground vehicles (AGVs) that provide high logistic flexibility and a reduced infrastructure footprint. However, many of such AGVs require additional infrastructure to travel along their predefined routes on the shop floor. This includes optical markers or guidance wires mounted on the floor. With industry pushing automation from mass products towards more flexible, changing production processes, the high set up costs for reconfiguring AGV routes tend to make automation of logistics with these platforms impractical. To make automation worthwhile in changing production

This work has partly been supported by the European Commission under grant agreement numbers FP7-248258-First-MM, FP7-260026-TAPAS, and FP7-248873-RADHAR.

Christoph Sprunk, Wolfram Burgard
Department of Computer Science, University of Freiburg, Germany,
E-mail: {sprunkc,burgard}@informatik.uni-freiburg.de

Boris Lau
E-mail: mail@borislau.de

Patrick Pfaff
KUKA Laboratories GmbH, Augsburg, Germany,
E-mail: patrick.pfaff@kuka.com

environments, mobile robots have to autonomously and accurately find their route to the designated goal.

Omnidirectional platforms can perform complex maneuvers in confined spaces due to their ability to drive in any direction. Therefore, we target this work towards omnidirectional robots. We present an all-around system for accurate autonomous navigation for omnidirectional robots that includes a robot localization module, a trajectory generation module and a robot control module. Our system is tailored to leverage the characteristics of omnidirectional robots, from path planning to trajectory execution. The core factors in industrial shop-floor navigation are navigation accuracy, safety, and efficiency. High navigation accuracy is a prerequisite for safety and means that the robotic platform will closely follow its planned motions. It needs to keep deviations from the planned behavior like cutting or overshooting corners to a minimum. For our system, we target deviations with respect to the global reference frame in the range of a few centimeters, which is well within the typical noise of global localization.

We address navigation accuracy by planning trajectories with velocities, accounting for platform constraints such as acceleration and maximum wheel velocities. Thereby, our system can accurately follow these planned trajectories. The trajectories are smooth and curvature continuous, this is a key factor for accurate execution and to reduce wear and tear on mobile robots that work on an industrial factory floor for extended periods of time. We address navigation safety by considering safety constraints for the trajectories. In particular, we impose obstacle-dependent speed limits. The closer the robot navigates to obstacles, the slower it is allowed to drive. This is an important behavior to avoid damages to the robot or its environment. Furthermore, driving slower near close obstacles enables higher accuracy in trajectory execution.

In large robotic platforms, like the one shown in Fig. 1, even low rotational velocities generate large tangential velocities of the robot chassis. In this work, we take into account the footprint of the robot chassis and constrain, during the motion planning, the maximum velocity achievable by any point on the chassis.

In addition to accuracy and safety, another objective of our system is navigation efficiency. Our system performs optimization of travel time while satisfying all of the above constraints. Our optimization procedure works by alternating adaptation of the path shape with recomputing the velocity profile. For increased efficiency, we also explicitly take into account the ability of omnidirectional robots to independently translate and rotate. It is important that autonomous robots that share the factory floor with human workers can cope with changing environments and unmapped obstacles. Therefore, our system regularly updates trajectories whilst the robot is moving. It smoothly joins the updated trajectory with the current trajectory and performs the necessary planning within a given short time deadline. For time efficiency in planning and optimization, we rely on a recursive collision checking approach for circular and rectangular elements. The approach leverages distance transforms and has not been reported in the literature to the best of our knowledge.

The system presented in this paper has been experimentally evaluated and quantified in all its key components showing high accuracy with efficient computation. We present also a real-world experiment consisting of almost 3 km navigation in 2 hours. Additionally, this work had direct impact in the industry and is currently embedded on the industrial mobile omnidirectional platform KUKA omniRob.

This manuscript is an extension of the work presented by Lau et al. [25] and Sprunk et al. [43]. We here describe the complete navigation system with a more detailed presentation of our trajectory generation and report additional extensive experiments.

The remainder of this paper is organized as follows. We first discuss related work in Sec. 2. Then, Sec. 3 presents an overview of the proposed system. We present our approaches for mapping and localization in Sec. 4. Sec. 5 describes the trajectory generation and optimization. The resulting trajectory is sent to the error feedback controller for execution, Sec. 6. We present the experimental evaluation of the system in Sec. 7.

2 Related Work

In this work we present a navigation system for omnidirectional robots in the context of industrial applications. Our trajectory generation module employs optimization techniques. We therefore discuss related work from the areas navigation systems, trajectory optimization, omnidirectional robots and industrial applications.

A successful navigation system for mobile robots from academia is the open source framework Carmen [6] initiated by Montemerlo et al. [31]. It localizes the robot in a grid map of the environment with the help of range sensors and Monte Carlo localization as proposed by Thrun et al. [47]. Carmen generates velocity commands for the robot by either following a global gradient field computed on the grid map or through waypoint following algorithms applied to a sequence of cells on the grid map.

The ROS navigation stack for the PR2 is a more recent system for which Marder-Eppstein et al. [29]

demonstrated robust performance. The system performs navigation in the plane but performs collision checks in 3D. It generates paths by an A* search on a costmap of the environment and uses the Dynamic Window Approach by Fox et al. [12] to generate velocity commands that follow the path to the goal. The system employs the adaptive Monte Carlo localization by Fox [11] which we also use in the system proposed in this paper.

Kümmerle et al. [23] present a navigation system for a differential drive robot in pedestrian zones that also performs traversability analysis of the environment. The system employs Monte Carlo localization on a grid map that it generates with a graph-based formulation of the simultaneous localization and mapping (SLAM) problem also employed in the proposed system. Since their application scenario is of larger scale, Kümmerle et al. propose to use tiled maps in their system. They realize navigation to a goal configuration with a three-tiered approach. On the highest level a topology planner operates on the map tiles while on the middle level a planner based on Dijkstra's algorithm generates waypoints within the map tiles. These waypoints serve as subgoals for a local planner that operates on a state lattice that includes the velocity of the robot in the plan [40].

The DARPA Grand and Urban Challenge triggered a considerable amount of research for navigation systems for autonomous cars [27, 30, 49, 54, 57, 58]. The systems developed for these challenges are tuned specifically to car-like vehicles driving within a given road network or maneuvering on a parking lot. Thrun et al. [49] present the system that won the Grand Challenge. They propose a separate velocity and steering controller to keep the robot at a lateral offset from the previously smoothed road description. The velocity controller slows the car down for lateral changes or when hitting bumps. Also the system for the Grand Challenge proposed by Ziegler et al. [58] and Werling and Gröll [54] relies on separate controllers for velocity and heading to track a geometric path.

Likhachev and Ferguson [27] present the system that won the Urban Challenge. The system generates curvature discontinuous geometric paths with Anytime Dynamic A* on a multi-resolution lattice with 32 discrete orientations. It tracks the path with a local planner and reduces the maximum speed for higher curvature sections to cope with the curvature discontinuities in practice.

While all of the above approaches determine velocities with controllers during execution, the systems for the Urban Challenge presented by Ziegler and Stiller [57] and Maček et al. [30] plan for velocities. Maček et al. [30] rely on partial motion planning that grows a tree of suitable velocity commands within a horizon and Ziegler and Stiller [57] use quintic splines to represent lane changing maneuvers in a spatio-temporal state lattice. Similar to Ziegler and Stiller we plan for velocities on paths based on quintic splines, however we consider an omnidirectional robot that is not restricted to lane changes.

Similar to our system, a number of approaches employ parametric path representations to inherently plan smooth paths. By modifying the parameters of such representations, one can deform these paths to account for obstacles [4, 7, 13, 24, 35, 56]. The elastic bands by Quinlan and Khatib [35] adjust such paths to evade dynamic obstacles using circular approximations of the free space around the robot. Brock and Khatib [4] adapted this idea to more complex robots with multiple degrees of freedom. Yang and Brock [56] extended the approach to elastic roadmaps that they deform to account for the motion of obstacles in the environment. Connors and Elkaim [7] proposed to model possibly colliding trajectories using splines and to establish collision-freeness by iteratively moving control points off the obstacles. However, their approach cannot guarantee to find a collision-free solution. Lamiraux et al. [24] also deform paths to evade obstacles, but specifically consider nonholonomic constraints. Fraichard and Delsart [13] also deform trajectories to restore collision-freeness after dynamic obstacles invalidated them.

There are also approaches that deform trajectories by optimization [5, 19, 36, 41]. Ratliff et al. [36] propose CHOMP, a gradient descent approach to compute collision free trajectories of a predetermined duration. They optimize with respect to the workspace integral of an obstacle distance dependent cost function. Byravan et al. [5] present T-CHOMP, an approach that extends this concept to time-dependent cost functions and also optimizes the timing of trajectories. Kalakrishnan et al. [19] propose a stochastic, gradient-free optimization to compute collision free trajectories for a fixed duration. Their cost function is a combination of smoothness, a minimum distance to obstacles, torque limits and constraints on the end effector pose. Schulman et al. [41] propose to find collision free paths with sequential convex optimization, repeatedly increasing penalties for violated constraints. In contrast to the above approaches our system starts optimization from a feasible initialization and uses the time of travel as cost function to balance curve radii, closeness to obstacles and arc length of the path while respecting constraints on velocity and acceleration.

Previous work on omnidirectional robots mostly covers fundamental control topics like position and velocity control [39, 53] and trajectory tracking [28]. Balkcom

et al. [2], Kalmár-Nagy et al. [20], as well as Purwin and D'Andrea [34] proposed approaches to determine short time-optimal trajectories without considering obstacles. These approaches can be applied to generate motion primitives or to implement smooth ad-hoc waypoint following. In this paper, we resort to a family of paths that we deform according to a cost function that also depends on the obstacles in the environment.

The work by Hornung et al. [18] for the omnidirectional PR2 robot employs Anytime Repairing A* search on a discrete set of motion primitives to generate smooth paths. As for most of the navigation systems discussed so far, these paths are executed by a trajectory rollout controller that follows their geometry but determines velocities ad-hoc when executing the path.

In the industrial context there are a variety of navigation systems for mobile robots available on the market today. The well-known approach by KIVA Systems consists of a fleet of mobile robots that fetch shelves to human workers to let them perform the necessary pick operations in warehouse automation. The robots navigate on a grid with the help of optical markers on the warehouse floor [16, 55]. Frog AGV Systems offers robots that rely on a grid pattern of magnets embedded in the shop floor for navigation [14]. There are also systems that do not rely on external infrastructure for navigation. The TransCar system sold by swisslog [46] is capable of laser-guided navigation in the context of hospital logistics. The BlueBotics ANT navigation system described by Tomatis [52] localizes itself with a Kalman Filter by matching previously mapped features that appear in laser scans of the environment. The system follows paths computed on a graph representation of the environment with the help of the Dynamic Window Approach by Fox et al. [12]. The ARNL navigation stack by adept mobilerobots [1] also uses this approach and computes the followed paths on a grid map built with range sensors.

Our system, instead, is an all-around robot navigation suite that includes mapping, localization, trajectory generation, and robot control. We tailored our system to exploit the capabilities of omnidirectional robots and to compute smooth trajectories that can be executed with high accuracy.

3 Overview

Our navigation system includes a mapping and localization module, a trajectory generation module and a motion execution module. For operation, the robot relies on a map of the environment. This is used for localization and robustly updated during operation (see Sec. 4). We outline the different steps and components used for computing and executing trajectories in Fig. 2.

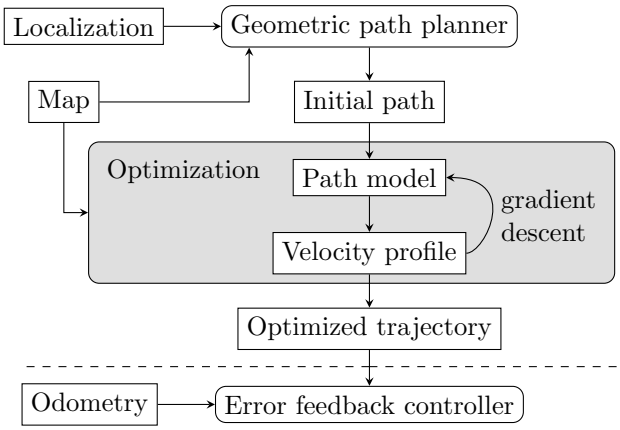

Fig. 2 Overview of our system for accurate navigation of omnidirectional robots. The dashed line marks the separation between trajectory generation and execution. The feedback loop for trajectory execution runs at a higher frequency than the (re-)planning of trajectories.

In a first step, our system employs a geometric path planner to find a collision free path from the current robot configuration to the goal (see Sec. 5.1). After expressing the initial path with our spline-based path model (see Sec. 5.3), an optimization procedure iteratively improves the shape of the path (see Sec. 5.5). We compute a velocity profile for a geometric path (see Sec. 5.4) that respects vehicle and safety constraints to generate a feasible time-parameterized trajectory suitable for accurate execution. The velocity profile also provides the time of travel of the current trajectory which is incorporated in the cost function for the optimization. Once the current trajectory cannot be further improved by the optimization or there is no time left, the optimized trajectory is sent to an error feedback controller for execution.

We decouple the generation of trajectories and their execution such that the feedback controller is executed at a higher frequency (e.g., 35 Hz) while updated trajectories are generated at a lower frequency (e.g., 1.5 Hz), see the dashed line in Fig. 2. This way, trajectory execution in the odometry frame of the robot can achieve high accuracy through a high-frequency feedback loop. Regular re-planning of trajectories in the global-frame accounts for accumulating drift in the odometry frame, localization noise and changes in the environment.

When our system plans an updated trajectory for the already moving robot, it first decides on the time frame within which the new trajectory should be ready. Then, using localization and the currently executed trajectory, it predicts the configuration of the robot at that point in time, including position, orientation, velocity, and curvature. To ensure a smooth transition to the new trajectory, we use this future configuration

of the robot as starting configuration of the robot in the new trajectory. Therefore, it is important that our path model supports setting the curvature at the start of the path, see Sec. 5.3.

4 The Mapping and Localization Module

The map used by our system consists in a 2D grid map of the environment. We employ a graph-based formulation of the simultaneous localization and mapping (SLAM) problem to build a grid map from sensor data as proposed by Dellaert and Kaess [8] and Grisetti et al. [15]. In this formulation, the poses of the robot are represented by nodes in a graph and edges represent constraints between poses. These originate either from wheel encoders or are extracted from sensor measurements. In our case, we compute incremental motion constraints from laser range readings with a scan matcher as detailed by Olson [33]. We detect constraints that correspond to re-visited places in the environment (loop closures) by employing FLIRT features as proposed by Tipaldi et al. [50, 51]. We optimize the resulting graph with the g2o framework by Kümmerle et al. [22].

For localization during operation, our system uses a Monte Carlo localization (MCL) approach as proposed by Dellaert et al. [9]. MCL uses a set of samples called particles to represent the belief of the system about its state. Whenever the robot moves, the approach propagates the particles according to the motion model of the robot. Whenever the robot takes a measurement of its environment, the particle weights are updated according to the likelihood of the measurement for the respective particle. In particular, our particle filter employs an odometry motion model and a beam-endpoint model, as defined by Thrun et al. [48]. The performance of MCL depends on the number of particles used to represent the belief. For efficiency and robustness, we apply the KLD sampling suggested by Fox [11]. This approach performs online adaptation of the number of employed particles to save computational resources meanwhile limiting the error introduced through the sample-based representation.

To account for dynamic changes in the environment, we continually update the map during robot motion. We use a robust and effective strategy that always combines the initial grid map with the latest range measurements. Based on the current pose of the robot, we set the grid cells that correspond to beam end points of the range measurements as occupied. When new range measurements are available, the changes that originated from the previous measurements are undone. We propagate these incremental grid map updates to other map representations used for efficient path planning and collision checking [26]: the distance map and the discretized Voronoi diagram of the map. The first represents the distance to the closest obstacle in each cell which we use for collision checking. The second provides a skeletonization of the environment that contains all points that are equidistant to at least two obstacles. We use the Voronoi diagram for planning of an initial path.

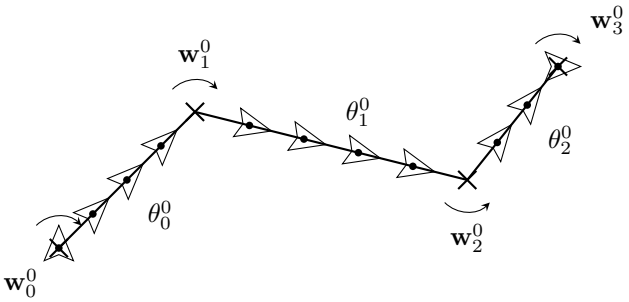

Fig. 3 Our system uses an initial path generated by a geometric path planner as initialization. The paths consist of translations with constant orientations θ_i^0 alternating with turns on the spot at the waypoints $\mathbf{w}_i^0$ of the initial path. The darts indicate the orientation of the robot.

5 The Trajectory Generation Module

This module is the backbone of the robot navigation system. It takes care of computing the trajectory shape and the velocity profile for the robot. This trajectory is ready to be executed by robot control. The trajectory generation module computes trajectories for omnidirectional robots and accounts for constraints to enable accurate and safe navigation while optimizing for time of travel.

5.1 Path Initialization

As shown in the overview in Fig. 2, we initialize the trajectory generation system with a path computed by a geometric path planner. For this step, our system is independent from the choice of the path planner engine.

The system accepts a path as a sequence of $M+1$ waypoints $\mathbf{w}_i^0$, $i \in \{0, \ldots, M\}$, connected by M straight line segments, see Fig. 3 for a visual explanation. The path is expected to be traversable without collisions when moving with constant orientation θ_i^0 between $\mathbf{w}_i^0$ and $\mathbf{w}_{i+1}^0$ for all segments. In this initial path, the robot changes its orientation by turning on the spot at the waypoints.

In practice, for path initialization, we use an efficient path planner that employs a discretized Voronoi diagram of the environment. In essence, the Voronoi

diagram is a graph that contains all points in the environment that are equidistant to at least two obstacles. The motivation behind our approach is to efficiently generate a conservative initialization for the path. Paths based on the Voronoi diagram are conservative in the sense that they emphasize distance to obstacles. Using such paths as initialization, our optimization-based trajectory generation can then decrease obstacle distance for more efficient paths. If however the planning time for trajectory optimization ends before the optimization converges to a balance between efficiency and safety, the resulting trajectory will be biased towards safety.

In a first step, our path planner connects start and goal to the discretized Voronoi diagram and searches the graph for a path that guarantees obstacle clearance for the incircle of the robot contour. Since the path at this point consists of a concatenation of grid cells, the planner converts it to straight line segments based on the Douglas-Peucker algorithm [17]. In the next stage, the planner checks for collisions when moving the robot along this path by following the orientation of its line segments and turning on the spot at their join points. Aligning the robot with the orientation of the line segments is the most conservative variant of augmenting the path with orientations, however one cannot guarantee the absence of collisions. In cases where the robot collides with the environment, an A* planner replaces the colliding segment of the path. This local A* planner accounts for a configuration space consisting of positions with orientations. Since the A* planner returns a sequence of grid cells, we again apply the Douglas-Peucker algorithm to convert the path into a sequence of orientation-augmented line segments. Here, in addition to the approximation error we also rely on collision checks when deciding whether or not to merge line segments in the Douglas-Peucker algorithm.

To retain completeness with this planning approach one can disconnect the corresponding edge of the discretized Voronoi diagram should the A* planner fail to resolve a collision for the respective segment and query the Voronoi diagram for an alternative path. Our path planner is similar in spirit to the one proposed by Foskey et al. [10], who bridge segments with a sampling-based planner in 3D workspaces.

5.2 Collision checking

In our system, not only the path initialization step but also other components of the trajectory generation make heavy use of collision checking. Therefore, we propose an efficient method for collision checking, shown in the pseudo code and the illustration in Fig. 4. The idea is to leverage an approximation of the shape of the robot and its payload as a set of circles and rectangles. These elements can be efficiently checked against a dynamically updated distance map $D(x, y)$ of the environment [26], which contains in each cell (x, y) the distance to the closest obstacle, $D(x, y)$. Circular elements with radius r and center (c_x, c_y) collide with an obstacle when $r > D(c_x, c_y)$. For rectangular elements centered at (c_x, c_y) we perform collision checking with a recursive procedure, see Fig. 4: If the obstacle distance $D(c_x, c_y)$ (dashed circle) is larger than the circumcircle of the rectangle (radius r_o), the rectangle is free of collisions. If $D(c_x, c_y)$ is smaller than the incircle of the rectangle (radius r_i), the rectangle collides. In the remaining case, i.e., $D(c_x, c_y)$ is in-between the incircle and the circumcircle, two rectangular areas can be recursively checked for collisions (R_1, R_2, shaded). For these, the dimensions and the center points can easily be determined.

```
collides_circ(circle C, distmap D)
  (c_x, c_y) ← center(C)
  return (D(c_x, c_y) ≤ radius(C))

collides_rect(rectangle R, distmap D)
  (c_x, c_y) ← center(R)
  if D(c_x, c_y) ≤ r_i then
    return true
  else if D(c_x, c_y) > r_o then
    return false
  compute rectangles R_1, R_2
  return (collides_rect(R_1, D)
          ∨ collides_rect(R_2, D))
```

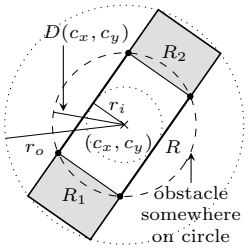

Fig. 4 Efficient collision checks for circular and rectangular elements. The drawing shows a rectangular element R with inner and outer diameter r_i and r_o. Only if the obstacle distance $D(c_x, c_y)$ at the center of R is between r_i and r_o, the result depends on recursive collision checks of R's subparts R_1 and R_2. Exceeding recursion can be terminated assuming collision.

In our system, we also require the distance map of the environment to determine speed limits for our trajectories that depend on obstacle distance, re-using computation. In case of highly complex robot shapes, it can still be beneficial to employ the incrementally updatable configuration space proposed by Lau et al. [26] for faster collision checks at the cost of an increased memory consumption.

5.3 Compact Path Model

In this section, we present the model we use to represent paths of the robot in the environment. After describing our compact path model in general, we provide details on how we transform the initial path given by the geometric path planner into our path representation in Sec. 5.3.3.

In practice, a path $\mathbf{s}(u) \in SE(2)$ is a progression of robot poses $\langle x, y, \theta \rangle$ in global coordinates as a function of an internal parameter $u \in [0, M] \subset \mathbb{R}, \quad M \in \mathbb{N}$.

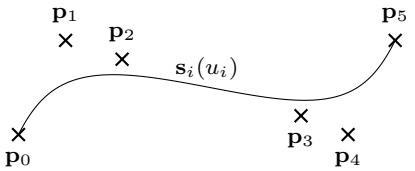

Fig. 5 A segment $\mathbf{s}_i$ of a quintic Bézier spline as used in our path model. The figure shows a 2D segment along with its control point $\mathbf{p}_0, \ldots, \mathbf{p}_5$. A Bézier segment passes through its first and last control point, first and second derivative at start and end can be influenced through the inner control points as shown in Eq. (2).

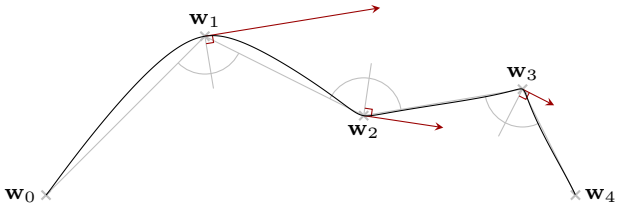

Fig. 6 Our path model sets direction of the first derivative at an inner waypoint to be orthogonal to the bisector of the angle formed by the adjacent waypoints (arrows). The magnitude of the first derivative is a free parameter and influences the wideness of curves.

To avoid confusion: the path $\mathbf{s}(u)$ determines where the robot drives in the environment but it does not determine *when* the robot arrives at a certain place and with which velocity. This is determined by velocity profiles which we introduce in Sec. 5.4.

The goal of our path representation is to model a path in the environment with a small set of parameters. With our representation, changes in these parameters result in modifications of the path and allow for fine adjustments of its shape. Furthermore, we enforce continuity in curvature for accurate and smooth execution.

We base our path model on quintic Bézier splines. We first discuss modeling the position of the robot, i.e., $\mathbf{s}(u) \in \mathbb{R}^2$ and then add orientation to our representation in Sec. 5.3.1, yielding $\mathbf{s}(u) \in SE(2)$. In the Bézier formulation, each segment $\mathbf{s}_i$ of a spline is a polynomial of degree five and it is defined for its parameter $u_i \in [0, 1]$ by six control points $\mathbf{p}_0, \ldots, \mathbf{p}_5 \in \mathbb{R}^2$:

$$\begin{aligned}\mathbf{s}_i(u_i) = &(1-u_i)^5\mathbf{p}_0 + 5(1-u_i)^4 u_i \mathbf{p}_1 + 10(1-u_i)^3 u_i^2 \mathbf{p}_2 \\ &+ 10(1-u_i)^2 u_i^3 \mathbf{p}_3 + 5(1-u_i) u_i^4 \mathbf{p}_4 + u_i^5 \mathbf{p}_5.\end{aligned} \tag{1}$$

As can be seen from Eq. (1), a segment interpolates between its control points, starting at $\mathbf{s}_i(0) = \mathbf{p}_0$ and ending at $\mathbf{s}_i(1) = \mathbf{p}_5$, see also Fig. 5. The spline segment is not passing through its intermediate control points. These directly and independently determine the first and second derivative of the segment at the start and end of the segment:

$$\begin{aligned}\mathbf{s}_i'(0) &= 5(\mathbf{p}_1 - \mathbf{p}_0), \quad \mathbf{s}_i''(0) = 20(\mathbf{p}_0 - 2\mathbf{p}_1 + \mathbf{p}_2), \\ \mathbf{s}_i'(1) &= 5(\mathbf{p}_5 - \mathbf{p}_4), \quad \mathbf{s}_i''(1) = 20(\mathbf{p}_3 - 2\mathbf{p}_4 + \mathbf{p}_5).\end{aligned} \tag{2}$$

We ensure continuity at the join points between individual path segments by requiring control points of neighboring segments to satisfy $\mathbf{s}_i(1) = \mathbf{s}_{i+1}(0)$. Similarly, we enforce constraints for the first and second derivative of the path and thereby achieve curvature continuity along the complete path. When concatenating the path $\mathbf{s}(u)$ from individual segments $\mathbf{s}_i(u_i)$, we map appropriately from the internal parameter u to the respective u_i. Through modification of $\mathbf{p}_1$ and $\mathbf{p}_2$ of the first segment, we have control over the first and second derivative at the start of the path. Thereby, we can control the curvature at the start of the path: a key property of our path model when planning updated trajectories.

Considering the continuity constraints introduced above, our path model has as free parameters: the position and first and second derivative at its start, end, and at the join points of path segments, which we call waypoints $\mathbf{w}_i$. In practice, we reduce the number of model parameters by using some heuristics. The first heuristic determines the direction of the first derivative at an inner waypoint $\mathbf{w}_i$ to be perpendicular to the angular bisector of the angle formed with the adjacent waypoints $\mathbf{w}_{i-1}, \mathbf{w}_{i+1}$, see the arrows in Fig. 6. The magnitude of the first derivative is a free parameter and represented through a scalar elongation factor e_i in the path model. As shown in Fig. 6, the elongation factor influences the radius of the curve performed at waypoint $\mathbf{w}_i$. The second heuristic determines the second derivative at inner waypoints. To achieve smooth curves we mimic the behavior of cubic splines which minimize the integral of the second derivative's absolute value on a segment. However, cubic splines connect with discontinuities in second derivative and hence in curvature at waypoints. To achieve a similar behavior with quintic splines, we set the second derivatives of the spline segments at a waypoint as a weighted average of the second derivatives that adjacent cubic spline segments would yield, if created at identical positions and same first derivatives at start and end of the segment. Fig. 7 visualizes the effect of the heuristic, it depicts the quintic spline resulting from our path model as well as the cubic spline used to compute the second derivative at the join point of the segments. The second derivative at an inner waypoint is given by a weighted average:

$$\begin{aligned}\mathbf{s}''(i) = &\frac{\|\mathbf{d}_{i,i+1}\|}{\|\mathbf{d}_{i-1,i}\| + \|\mathbf{d}_{i,i+1}\|} \lim_{u \to i^-} \mathbf{s}''_{\text{cubic}}(u) \\ &+ \frac{\|\mathbf{d}_{i-1,i}\|}{\|\mathbf{d}_{i-1,i}\| + \|\mathbf{d}_{i,i+1}\|} \lim_{u \to i^+} \mathbf{s}''_{\text{cubic}}(u),\end{aligned} \tag{3}$$

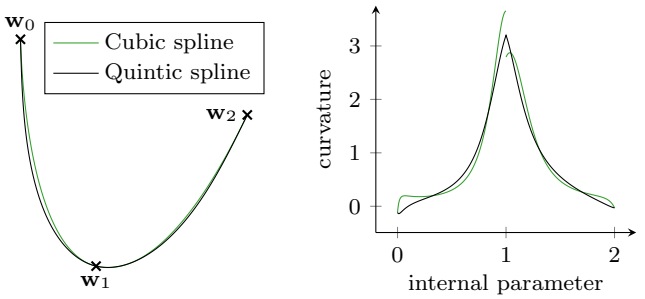

Fig. 7 Our path model employs quintic splines and determines their second derivative at inner waypoints with a heuristic: We choose a weighted average of the discontinuous second derivative of a cubic spline constructed with matching waypoints and first derivatives. This way, we maintain curvature continuity while mimicking cubic splines.

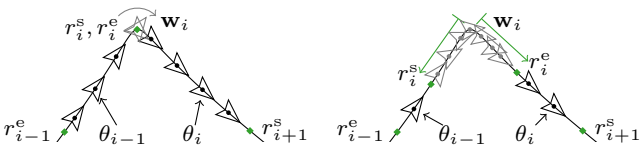

Fig. 8 Influencing simultaneous rotation and translation with rotational-control-points (green diamonds): r_i^{s} and r_i^{e} coinciding with waypoint $\mathbf{w}_i$ yields a turn on the spot (left). If r_i^{s} and r_i^{e} move away from $\mathbf{w}_i$ rotation takes place simultaneously with translation (right). The darts indicate robot motion with constant (black) and changing (gray) orientation.

where $\mathbf{d}_{i,i+1} = \mathbf{w}_{i+1} - \mathbf{w}_i$, and $\mathbf{s}_{\mathrm{cubic}}$ is a cubic spline constructed with matching waypoints and first derivatives.

So far, we discussed how our path representation explicitly models the position in the environment. The orientation of the robot could be derived from the first order derivative of the path. This would be a model suitable for differential drive and synchro-drive robots where the first derivative $\mathbf{s}'(u)$ of the path uniquely determines the orientation of the robot as $\theta(u) = \mathrm{atan2}(\mathbf{s}'_y(u), \mathbf{s}'_x(u))$. However, we want to fully leverage with our model the capabilities of omnidirectional robots that can rotate independently from their translational motion. Thus, we explicitly represent the orientation of the robot as an independent component in our path model.

5.3.1 Modeling independent rotation and translation

To explicitly represent the orientation of the robot, we add a corresponding dimension to the control points of the quintic Bézier segments, such that $\mathbf{p}_0, \ldots, \mathbf{p}_5 \in SE(2)$ in Eq. (1). At first glance, augmentation of the waypoints with a dedicated dimension would suffice to represent orientation independently from translation. However, the robot would continuously rotate between two waypoints as the connecting spline segment interpolates between the orientations of adjacent waypoints. Recall that the representation of the initial path provided by a geometric planner (see Sec. 5.1) intends for the robot to travel with constant orientation on a segment and to perform rotations on the spot only at the waypoints. We could model rotations on the spot by inserting two waypoints with different orientations at the same position. However, we also want to model simultaneous rotation and translation and to smoothly transition to such behavior from the initial path. Therefore, we introduce a more flexible concept in our representation and expose parameters to control the transition to simultaneous rotation and translation. To "distribute" the rotation on the spot from a waypoint over the adjacent segments, we introduce rotational-control-points r_i^{s} and r_i^{e} on the incoming and outgoing segments connected to each waypoint $\mathbf{w}_i$. These points mark the beginning (r_i^{s}) and the end (r_i^{e}) of the robot rotation from θ_{i-1} associated with the incoming segment of $\mathbf{w}_i$ to θ_i related to the outgoing segment. The idea is visually exemplified in Fig. 8: On the left, there is a configuration where r_i^{s} and r_i^{e} coincide with $\mathbf{w}_i$, resulting in a rotation on the spot. On the right, there is a configuration where r_i^{s} and r_i^{e} are away from $\mathbf{w}_i$ (green arrows), this causes a smooth rotation during translation. When r_i^{e} of waypoint $\mathbf{w}_i$ coincides with r_{i+1}^{s} of $\mathbf{w}_{i+1}$, this results in an uninterrupted rotation on the segment.

Technically, rotational-control-points are realized by subdividing the spline segments in a way that leaves the positional part unaffected and affects only orientation to achieve the desired behavior. For this, we evaluate the spline segment with its derivatives at the subdivision point and use these values as end and start parameters at the new join point. Due to the change in parameterization, the derivatives need to be scaled as follows. Suppose spline segment i, $i \in \{0, \ldots, M-1\}$ is subdivided at internal parameter $u_d \in (i, i+1)$. The new sub-segments span the entire domain of the internal parameter: $[i, i+1]$ and $[i+1, i+2]$ instead of $[i, u_d]$ and $[u_d, i+1]$. To retain the shape of the spline segments the k-th derivative of the first sub-segment has to be scaled by $(u_d - i)^k$ and the k-th derivative of the second sub-segment by $(i + 1 - u_d)^k$.

When subdividing segments at rotational-control-points, the orientation component of the first and second derivative is set to zero, as these points mark a transition between a segment with constant orientation and a segment with varying orientation. At the waypoints, we set the first derivative to point into the direction of rotation, $T_{i,\theta} = \frac{1}{2} e_\theta \, (\theta_i - \theta_{i-1})$, where e_θ is a parameter of the path model for each waypoint. The second order derivative of the orientation component follows

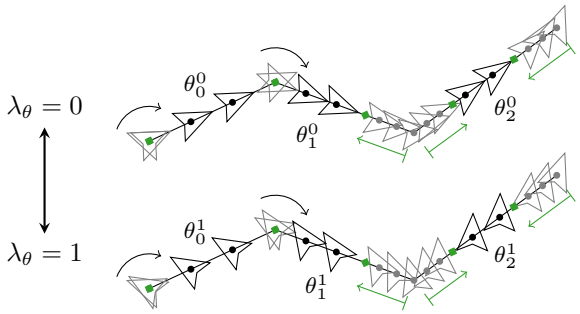

Fig. 9 Paths with different rotation behaviors. *Top:* initial paths use $\lambda_\theta = 0$ to orient the robot as specified by the waypoint planner. *Bottom:* minimized change of orientation with $\lambda_\theta = 1$. The optimization can adjust λ_θ to interpolate between these two extremes. For reference, the figure also shows the location of rotation control points (green diamonds).

the same heuristic as its translational counterpart at every waypoint.

The employed spline segments interpolate between the orientations at rotational-control-points and the waypoints. To prevent unintended turns into the wrong direction, in practice we change the space for the robot path from $SE(2)$ to $\mathbb{R}^3$ and add appropriate multiples of 2π to the orientation values. For example, we replace a turn $(\pi-\epsilon) \rightarrow (-\pi+\epsilon)$ with the turn $(\pi-\epsilon) \rightarrow (\pi+\epsilon)$ for $0 < \epsilon < \pi$ to model paths with small turns that cross the $-\pi/\pi$-border.

5.3.2 Influencing the amount of rotation along the path

The rotational-control-points r_i^{s} and r_i^{e} introduced above determine *where* on the path the robot rotates. However, the initial values θ_i^0 for the constant orientation θ_i between waypoints given by the geometric path planner can be conservative. Adjusting the θ_i to reduce the total amount of rotation can reduce travel time or energy consumption. In the example in Fig. 9, the orientation behavior given by the initial path requires the robot to drive forward on every segment of the path (top). The bottom part of the figure shows an orientation behavior that minimizes the total amount of rotation. Here, we introduce a parameter for our path model to smoothly change between the two extremal orientation behaviors.

Usually, the orientation of the robot is predetermined at start and goal. For a path with waypoints $\mathbf{w}_i$, $i \in \{0, \ldots, M\}$, we compute the orientations θ_i^1 that correspond to a minimal change of orientation. They interpolate the given start and end orientation θ_{start} and θ_{end} of the path, as shown in Fig. 9 (bottom). They are determined according to

$$\theta_i^1 = \theta_{\mathrm{start}} + f_i \, (\theta_{\mathrm{end}} - \theta_{\mathrm{start}}) \ . \tag{4}$$

Here, $f_i \in [0, 1]$ stands for the fraction of path length up to the middle of the i-th segment, and is computed as

$$f_i = \frac{\sum_{k=1}^{i} \|\mathbf{w}_k - \mathbf{w}_{k-1}\| + \frac{1}{2}\|\mathbf{w}_{i+1} - \mathbf{w}_i\|}{\sum_{k=1}^{M} \|\mathbf{w}_k - \mathbf{w}_{k-1}\|} \ . \tag{5}$$

We introduce a parameter $\lambda_\theta \in [0, 1]$ that blends the rotational behavior between the initial (θ_i^0) and minimized (θ_i^1) values according to

$$\theta_i = (1 - \lambda_\theta)\theta_i^0 + \lambda_\theta \theta_i^1 \ . \tag{6}$$

For $\lambda_\theta = 0$, the initial θ_i^0 orientations are used as shown in Fig. 9 (top). Choosing $\lambda_\theta = 1$ uses the θ_i^1 and thus achieves a minimal change of rotations as depicted in Fig. 9 (bottom) which however might not be free of collisions. The optimization starts with $\lambda_\theta = 0$ and continuously changes this parameter to obtain more efficient paths that are still collision-free. Note that this parameter does not affect whether rotations occur on the spot or while translating. In Fig. 9 (top vs. bottom), compare the black darts (constant orientation) and gray darts (changing orientation): their position along the path is identical, only their heading differs. The orientation component $\mathbf{w}_{i,\theta}$ of a waypoint $\mathbf{w}_i$ is set to the average value of the adjacent segment orientations, $\mathbf{w}_{i,\theta} = \frac{1}{2}(\theta_{i-1} + \theta_i)$.

For longer paths, adjusting the overall orientation behavior for the whole path with only one parameter might not be feasible. A narrow passage along the path might constrain the orientation of the platform and thereby prevent any adaptions for the rest of the path. For such cases, once can split the path and insert individual λ_θ for the sub-paths to enable decoupled adjustment of rotational behavior. In practice, we do not apply this technique since we limit optimization of longer paths to the fraction within the field of view of the robot's sensors, see also Sec. 7.

5.3.3 Modeling the initial path

As described in Sec. 5.1 and shown in Fig. 2 we initialize our system with a path generated by a geometric path planner that consists of straight-line motions alternating with turns on the spot. To pass this initial path to the optimization, we transform it into an instantiation of our compact path model.

The geometric path planner provides a path to the goal that is free of collisions. To ensure a feasible starting point for the optimization we need to set the parameters of our path model in a way that closely approximates the initial path. We set the waypoints $\mathbf{w}_i$ of our path model to the points provided by the path planner, $\mathbf{w}_i = \mathbf{w}_i^0$. By selecting low values for the elongation factors e_i we achieve a close approximation of a straight-line path with sharp curves. One can control the approximation

error by subdividing longer segments with additional waypoints. To correctly model the orientation behavior, we set the rotational-control-points to coincide with their waypoints for turning on the spot (see Sec. 5.3.1) and set all $\lambda_\theta = 0$ to enforce the segment orientations given by the path planner (see Sec. 5.3.2). The initial scaling factor for the orientational component of the first derivative at waypoints is set to $e_\theta = 1$ for the initial path. For an example of an initial path represented by our model, see the top row of Fig. 10.

Starting from the initial path, the optimization moves inner waypoints to adapt the path to the environment, widens curves at waypoints by adapting e_i, distributes rotation along the path by moving r_i^{s} and r_i^{e} away from $\mathbf{w}_i$ and changes the amount of rotation performed along the path by adapting λ_θ. The resulting changes in path shape influence how fast the robot can travel along the path to the goal. The velocities along the path and thereby the total time of travel are determined by the velocity profile which we present in the next section.

5.4 Velocity Profiles

As introduced in the previous section, a path $\mathbf{s}(u)$ defines a progression of poses for the robot from start to goal. Together with a mapping $u(t)$ from time t to the internal path parameter u, the path becomes a trajectory that defines the configurations of the robot over time t. Thus $u(t)$, which is strictly monotonically increasing, defines a velocity profile and the time t_{travel} required to execute the trajectory. For a path $\mathbf{s}(u)$ defined by the parameters described in Sec. 5.3, we seek to compute the mapping $u(t)$ that corresponds to the fastest velocity profile that respects robot and safety constraints.

The speed at which the robot traverses a path $\mathbf{s}(u)$ is determined by the mapping $u(t)$, such that $\mathcal{T}(t) := \mathbf{s}\left(u(t)\right)$ determines the pose and velocities of the robot over time t. A trajectory $\mathcal{T}(t)$ defines velocities by its first derivative with respect to time t,

$$\dot{\mathcal{T}}(t) = \mathbf{s}'(u(t))\,\dot{u}\,(t)\,, \tag{7}$$

where the dot and the prime denote derivatives with respect to t and u, respectively.

We finely discretize $\mathbf{s}$ into a set of points, e.g., whenever the robot would travel more than 0.02 m or rotate more than 0.02 rad. For these points we compute the maximum $\dot{u}(t)$ that satisfies a set of constraints. These constraints do not necessarily correspond to physical limitations of the robot but are rather set to more conservative values to reduce wear and tear and ensure safety and comfort for platform load as well as workers sharing the shop floor.

5.4.1 Maximum velocity

The translational and rotational velocities of the platform can be limited to $v_{\max}, \omega_{\max}$. The resulting constraint for $\dot{u}(t)$ is given by

$$\dot{u}(t) \leq \frac{v_{\max}}{\|\mathbf{s}'_{xy}(u(t))\|}, \quad \dot{u}(t) \leq \frac{\omega_{\max}}{|\mathbf{s}'_\theta(u(t))|}. \tag{8}$$

5.4.2 Obstacle-dependent velocity limit

For safe navigation behavior, we impose an obstacle-dependent speed limit on the robot. After determining the distance of the robot contour to the closest obstacle, we limit its velocity to an upper bound that allows coming to a complete halt before collision with an obstacle. The computation assumes a constant acceleration and allows to parameterize a reaction time and a maximum deceleration for these braking maneuvers.

5.4.3 Maximum velocity of robot contour

When navigating with larger robots and in workspaces that are shared with humans it can be beneficial to not only limit the velocity of the robot at its center of motion as introduced above but also account for the effect of rotational velocity at the contour of the robot: we limit the maximum translational speed of *any* point of the robot with respect to the environment.

If the center of motion of the robot moves with velocity $v = (v_x, v_y, v_\theta)^\top$, a point $p = (p_x, p_y)^\top$ on the robot expressed relative to the robot's center of motion moves with velocity

$$v_p = (v_x - v_\theta p_y, v_y + v_\theta p_x, v_\theta)^\top. \tag{9}$$

To enforce a maximum translational velocity of a point p on the robot, we require

$$\|v_p\| = \sqrt{(v_x - v_\theta p_y)^2 + (v_y + v_\theta p_x)^2} \leq v_{\max}. \tag{10}$$

We obtain the following constraint by substituting $v = (v_x, v_y, v_\theta)^\top = \mathbf{s}'(u(t))\,\dot{u}(t)$ into Eq. (10):

$$\dot{u}(t) \leq \frac{v_{\max}}{\sqrt{(\mathbf{s}'_x - \mathbf{s}'_\theta p_y)^2 + (\mathbf{s}'_y + \mathbf{s}'_\theta p_x)^2}}, \tag{11}$$

where for readability $\mathbf{s}'$ is short for $\mathbf{s}'(u(t))$. For every discretized path point, the point p of the robot contour that maximizes the denominator of Eq. (11) gives the strongest constraint on $\dot{u}(t)$. For a rectangular robot with width w and length l, we can exploit the symmetry to formulate a single constraint:

$$\dot{u}(t) \leq \frac{v_{\max}}{\sqrt{(|\mathbf{s}'_x| + |\mathbf{s}'_\theta| l/2)^2 + (|\mathbf{s}'_y| + |\mathbf{s}'_\theta| w/2)^2}}. \tag{12}$$

5.4.4 Wheel turn rate

In addition to limiting the resulting velocity of the robot, it can also be necessary to account for a maximum wheel turn rate to model friction or motor saturation effects that would prevent the accurate execution of feasible trajectories. A good example are differential drive robots that have limited wheel turn rates: the maximum achievable translational velocity depends on the radius of the driven curve.

Let $\psi_i(u)$ be the turn rate of wheel i computed from $\mathbf{s}'(u)$. To respect a maximum turn rate $\psi_{\max}$, we need to constrain $\dot{u}(t)$ by

$$\dot{u}(t) \leq \frac{\psi_{\max}}{\max\limits_{i \in \text{wheels}} \{\psi_i(u(t))\}}. \tag{13}$$

For an omnidirectional robot with omniWheels, the turn rates of the wheels $\psi_i(u)$ can be computed from the orientation $\mathbf{s}_\theta(u(t))$ and the path derivative $\mathbf{s}'(u(t))$. After rotating $\mathbf{s}'(u(t))$ by $-\mathbf{s}_\theta(u(t))$ into the robot frame, the turn rates can be retrieved by application of the equations of motion, see Eq. (16) together with Fig. 11 and the explanation in Sec. 6 for the ones of the robot used in our experiments.

5.4.5 Centripetal acceleration

To prevent skidding and protect sensitive payload, a maximal centripetal acceleration a_c can be enforced by

$$\dot{u}(t) \leq \sqrt{\frac{a_\text{c}}{\|\mathbf{s}'_{xy}(u(t))\| \cdot |\mathbf{s}'_\theta(u(t))|}}. \tag{14}$$

We enforce all of the above constraints at each discretized path point, i.e., we set the maximum $\dot{u}(t)$ that satisfies all constraints. This implicitly defines the maximum translational and rotational velocity at each support point (this is called velocity limit curve in literature). In a second step, we further decrease $\dot{u}(t)$ at the support points to also respect acceleration constraints.

5.4.6 Acceleration

We assume constantly accelerated motion of the robot between the discretized path points. To ensure safe transportation of sensitive payload, we limit the range of allowed acceleration and deceleration in the intervals between the path points. To determine the minimum and maximum velocities achievable by limited acceleration, we compute the arc length and orientation distance of the respective intervals by numerical integration.

We reduce the values for $\dot{u}(t)$ in two passes originating from beginning and end of the trajectory to ensure that the acceleration constraints are met between the supports points. The procedure is very similar to the time-scaling algorithm proposed by Bobrow et al. [3] and Shin and McKay [42]. Given $\dot{u}(t)$, we can now compute the mapping from time to internal parameter $u(t)$.

5.5 Optimization

We employ optimization to improve the initial trajectory with respect to a user defined cost function, e.g., time of travel. As shown in Fig. 2, the optimization is initialized with the initial path (see, Sec. 5.3.3) and then iterates between two steps. First, it computes a velocity profile for the path described by the current parameters. Then, it uses the inferred time of travel to compute the cost of the current trajectory. The change in cost then determines how to change the path parameters for the next iteration via magnitude-free gradient descent.

The optimization influences the shape of the trajectory through modification of the parameters of the compact path model, see Sec. 5.3. In particular, the effect of the optimization is the widening of the sharp turns of the initial path to a point where the increased length of the path balances with the higher admissible velocities in curves with larger radius. The optimization also balances path length and distance to obstacles that impose velocity limits on the robot. Here, the optimization changes the location of inner waypoints in a coordinate system that is oriented along the gradient of the distance-transformed map. This means that the two degrees of freedom for moving the waypoint correspond to getting closer or farther away from the nearest obstacle and to an orthogonal movement with respect to it. Finally, the optimization also gradually changes the orientation behavior of the initial path to a simultaneous translation and rotation. Additionally, it reduces the overall robot rotation in the path, as long as this reduces the time of travel.

We propose an optimization procedure that is based on the update rule of RPROP [37], which is a derivative-free optimization algorithm proposed by Riedmiller and Braun and known for its robust convergence. The proposed procedure is shown as pseudo-code in Alg. 1.

In lines 1–3 the algorithm sets as starting point the trajectory generated from the initial path and resets the step size Δ_p for all parameters p that are to be optimized. As long as planning time is left and non-negligible improvements occur for the cost of the trajectory (lines 4, 5, 23) the algorithm iterates through the list of tunable path model parameters. First, the currently selected parameter is changed according to its current associated step size Δ_p and then the trajectory is updated accordingly (lines 12, 13). Then, we adapt the step size for the current parameter depending on

Algorithm 1 Trajectory optimization with respect to the user-defined cost function c.

1: $\mathcal{T}_{\text{best}} \leftarrow$ initial trajectory
2: $\mathcal{P} \leftarrow$ parameters of initial trajectory
3: $\Delta_p \leftarrow \Delta_p^0 \quad \forall p \in \mathcal{P}$ // reset step size for all parameters
4: **repeat**
5: $\quad \Delta_c \leftarrow 0$ // track cost improvement
6: $\quad$ **for all** $p \in \mathcal{P}$ **do**
7: $\quad\quad \mathcal{T}_{\text{curr}} \leftarrow \mathcal{T}_{\text{best}}$
8: $\quad\quad \Delta_p \leftarrow \Delta_p^0$ // always reset step size
9: $\quad\quad i \leftarrow 0$ // track number of iterations
10: $\quad\quad$ **repeat**
11: $\quad\quad\quad i \leftarrow i + 1$
12: $\quad\quad\quad p \leftarrow p + \Delta_p$
13: $\quad\quad\quad \mathcal{T}_{\text{mod}} \leftarrow \text{modifyPathParameter}(\mathcal{T}_{\text{curr}}, p)$
14: $\quad\quad\quad \Delta_p \leftarrow \begin{cases} 1.2\Delta_p & \text{c}(\mathcal{T}_{\text{mod}}) < \text{c}(\mathcal{T}_{\text{curr}}), \\ -0.5\Delta_p & \text{else} \end{cases}$
15: $\quad\quad\quad \mathcal{T}_{\text{curr}} \leftarrow \mathcal{T}_{\text{mod}}$
16: $\quad\quad\quad$ **if** $\text{c}(\mathcal{T}_{\text{mod}}) < \text{c}(\mathcal{T}_{\text{best}})$ **then**
17: $\quad\quad\quad\quad \Delta_c \leftarrow \max\{\Delta_c, \text{c}(\mathcal{T}_{\text{best}}) - \text{c}(\mathcal{T}_{\text{mod}})\}$
18: $\quad\quad\quad\quad \mathcal{T}_{\text{best}} \leftarrow \mathcal{T}_{\text{mod}}$
19: $\quad\quad\quad\quad$ **break** // continue with the next parameter
20: $\quad\quad$ **until** $|\Delta_p| < \epsilon_p \vee i > i_{\max} \vee$ planning time is up
21: $\quad\quad$ **if** $|\Delta_p| < \epsilon_p \vee i > i_{\max}$ **then**
22: $\quad\quad\quad \Delta_p \leftarrow \Delta_p^0$ // reset step size when stuck
23: **until** $\Delta_c < \epsilon_c \vee$ planning time is up
24: **return** $\mathcal{T}_{\text{best}}$

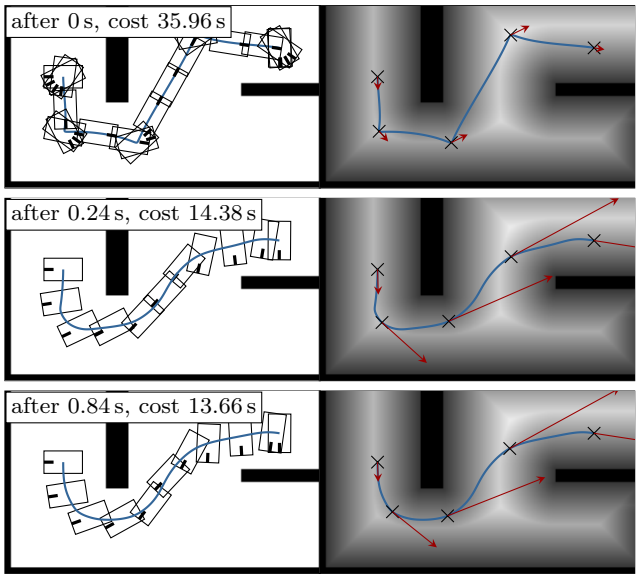

Fig. 10 Progress of the trajectory optimization over time. The optimization balances path length, distance to obstacles, orientation and trajectory shape for faster traversal. The left part of the figure shows the current trajectory with overlays of the robot contour in the grid map of the environment. The right part shows the trajectory with waypoints $\mathbf{w}_i$ (crosses) and the first derivatives of the path (arrows) on a distance transform of the environment. The figure shows obstacles in black and darker cells are closer to obstacles.

whether or not the modified trajectory has a lower cost than the previous one (line 14). We do this according to the update rule of RPROP: If the change of parameter p resulted in an improvement, the step size is increased by a factor of 1.2, otherwise, we halve the step size and invert the direction of the step. If at any point in this procedure an improvement is made over the currently best trajectory, the cost improvement is recorded for the abortion criterion (line 17) and the corresponding parameters are saved (line 18). To avoid local minima, the optimization also continues with the next parameter in the list as soon as an improvement has been made (line 19). In the experiments, we compare versions with and without this continuing. The optimization of an individual parameter is also aborted once its step size falls below a threshold or a maximum number of iterations is exceeded (line 20). In the experiments, we also compare several strategies: always resetting the parameter step size (line 8), never resetting it, or resetting it only when optimization is stuck (line 21, 22). All lines that are relevant for this comparison are shaded in Alg. 1.

Fig. 10 shows the progress of the optimization for an exemplary trajectory. To achieve faster travel times the optimization increases the magnitude of the tangents to widen the curves. It furthermore balances the path length and the distance to obstacles and changes the overall orientation behavior along the path.

During computation of the velocity profile we also check for collisions of the robot with the environment. Trajectories that collide with the environment are assigned infinite cost. At each iteration our method returns feasible trajectories with decreasing cost. As obstacles that cause collisions repel the optimization via infinite cost, the optimization cannot "jump" over obstacles.

When planning a trajectory for a robot that is already in motion, the velocities of the robot at the start of the trajectory are non-zero. In this context, it can occur that the shape of the initial trajectory prevents a transition with continuous velocities, e.g., due to a sharp curve near its beginning that enforces a low robot velocity. In such situations, we change the cost function of the optimization to minimize the amount of velocity discontinuity between desired and maximum achievable velocities at the start of the new trajectory. In this way, the optimization adapts the shape of the trajectory to allow a smooth fit. As soon as parameters have been found that match the start velocities, the optimization switches back to the original cost function for the remaining planning time. As the mentioned discontinuities are mostly fixed by increasing the magnitude of the first derivative of the starting waypoint, we do not continue with the next free parameter upon improvement.

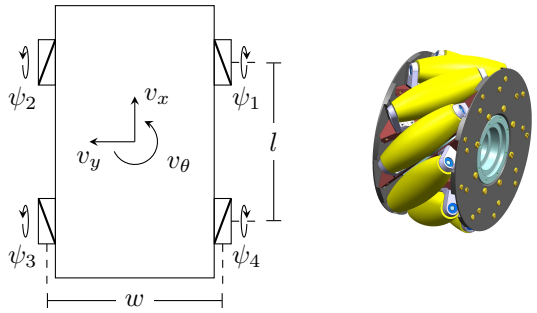

Fig. 11 Schematic drawing of the robot used in our experiments. The robot is driven by four Mecanum wheels shown on the right. Their passive rollers are oriented at $\pm\frac{\pi}{4}$, the respective orientation on the floor is shown by diagonal lines in the drawing on the left.

Fig. 12 The mobile manipulator omniRob by KUKA that we used in our experiments.

6 The Robot Control Module

As shown in the lower part of Fig. 2, the final optimized trajectory is sent to an error feedback controller for execution. Before execution, we transform the trajectory from global map coordinates into the odometry frame of the robot. The error feedback controller then executes the trajectory solely based on odometry information, decoupled from global trajectory planning and global localization. Thereby, the error feedback controller can run at a higher frequency than the trajectory planning. We account for accumulating odometry drift and changes in the environment by planning updated trajectories while the robot executes the trajectory.

The input for the controller are the planned position $\mathcal{T}(t)$ and velocities $\dot{\mathcal{T}}(t)$ for the current time t as determined by the trajectory $\mathcal{T}$ as well as the current position $\mathbf{o}(t)$ in the odometry frame as measured by the robot's wheel encoders. The output of the controller is a velocity $v = (v_x, v_y, v_\theta)^\top$ that is sent as a command to the robot. The controller computes the command velocity as follows:

$$v = \dot{\mathcal{T}}(t + \Delta t_{\text{del}}) + \text{diag}(g_t, g_t, g_r)\,(\mathcal{T}(t) - \mathbf{o}(t))\,, \quad (15)$$

where g_t and g_r are the translational and rotational gain, respectively. The control law consists of two summands, the first summand is the feed-forward part and the second summand is the error feedback. The feedforward part is the velocities as planned in the trajectory. We retrieve them for a point in time that is shifted by Δt_{del} into the future. This accounts for the time delay that occurs before the robot is actually executing the command.

For execution by the robot, the commanded velocity needs to be transformed into wheel turn rates. These are retrieved from the equations of motion for the particular robot used. For the omnidirectional robot used in our experiments, the equations of motions follow [32, Eq. (6.2.13)]:

$$\begin{pmatrix}\psi_1\\ \psi_2\\ \psi_3\\ \psi_4\end{pmatrix} = \frac{1}{r}\begin{pmatrix}1 & 1 & 0.5\,(w+l)\\ 1 & -1 & -0.5\,(w+l)\\ 1 & 1 & -0.5\,(w+l)\\ 1 & -1 & 0.5\,(w+l)\end{pmatrix}\begin{pmatrix}v_x\\ v_y\\ v_\theta\end{pmatrix}, \quad (16)$$

where r is the wheel radius and w and l are track and wheelbase, respectively. Fig. 11 shows a schematic drawing that introduces the used quantities. The above equation is also used when computing constraints regarding the maximum turn rate of wheels, see Sec. 5.4.

7 Evaluation

We evaluate the core navigation components of our system: the trajectory generation module and the robot control module. We start by evaluating the influence of the initial path planner on the proposed method. Then, we analyze our trajectory optimization by comparing the results of several variants on a set of navigation tasks. On these same set of tasks we also compare the trajectories of our system with the ones generated by an approach based on the RRT* path planning algorithm.

We present real-world experiments for evaluating the robot control module. We examine the influence of different constraint settings on the system behavior. Then, we present an evaluation of the system capability to follow the planned trajectories in a changing environment. These experiments have been conducted with the KUKA omniRob platform shown in Fig. 12. The omnidirectional platform has dimensions of ca. $0.7\,\text{m} \times 1.2\,\text{m}$ and perceives its environment with two SICK laser range finders mounted in a way that yields a 360 degree view. We performed all computations on a consumer grade notebook that was mounted on the platform.

When performing trajectory planning in real-world changing environments, it is not economical to spend computational resources on optimizing trajectories all the way to the goal pose for longer navigation tasks. The

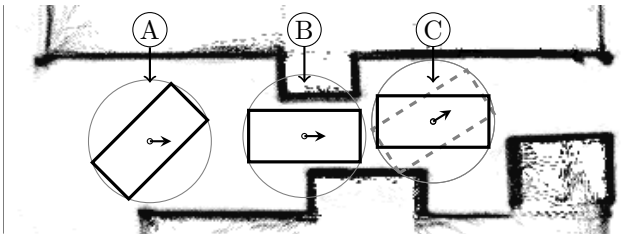

Fig. 13 Omnidirectional platforms are versatile in narrow spaces. They can rotate freely in open spaces (A), move with constant orientation in narrow passages (B), and execute maneuvers (C) where non-holonomic trajectories would lead to collisions (dashed). For readability, the figure contains the circumcircle of the rectangular robot.

later parts of a longer trajectory are outside the sensors' field of view and the environment is likely to change until the robot arrives there, requiring to update the trajectory, discarding the initial results. We also require regular updating of planned trajectories to cope with localization errors and accumulating odometry drift. In our experiments we found a good trade-off by using the first four waypoints of the initial path for trajectory planning and triggering replanning of updated trajectories every 0.6 s, which is the time allotted for path planning and trajectory optimization.

At the end of this section, we report on real-world robotic applications of this work.

7.1 Influence of the Choice of Geometric Path Planner

Our system uses an initial path generated by a geometric global path planner to generate an initial trajectory as initialization for the optimization. Naturally, the choice of planning algorithm can have an influence on the final result.

2D path planners ignore orientation and therefore require the circumcircle of the robot to be obstacle-free along the path, see for example Fig. 13 (A). Waypoint planners that account for the orientation and shape of the robot during forward motion are able to find paths through passages as shown in Fig. 13 (B). Planners operating in the full holonomic configuration space (C-space) of the robot can also find paths through passages that require lateral or diagonal movements as in Fig. 13 (C). Our approach can generate admissible spatio-temporal trajectories for all these situations, given a suitable waypoint planner.

To assess the robustness of our system against the choice of waypoint planner, we compare the optimization results for the Carmen 2D value iteration planner, an A*-based C-space planner, and our hybrid approach that uses Voronoi diagrams where possible and resorts to C-space planning when necessary (see Sec. 5.1). We

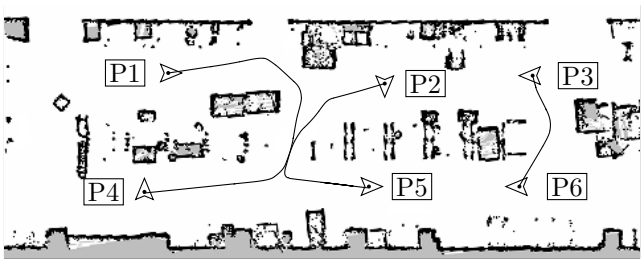

Fig. 14 Map with start/goal poses used in our evaluation of different path planners in combination with our trajectory generation method. The figure also contains selected example trajectories between the poses.

Table 1 Numerical results of global path planner comparison.

Planner	t_{plan}	t_{opt}	$t_{\text{travel-I}}$	$t_{\text{travel-O}}$
2D value iteration	0.022 s	0.55 s	82.37 s	38.11 s
Voronoi	0.009 s	0.46 s	36.28 s	28.46 s
configuration-space	4.864 s	0.43 s	31.49 s	27.19 s

selected six start/goal poses on a map of a factory floor as shown in Fig. 14 and let a simulated robot perform travel tasks for all of the 30 start-goal combinations.

Tab. 1 shows the average times needed for planning (t_{plan}), optimization (t_{opt}), and execution of the initial and optimized trajectories ($t_{\text{travel-I}}$ and $t_{\text{travel-O}}$, respectively). As expected, the C-space planner requires substantially more time to find an initial path compared to the others, which prevents its use in most practical applications. On the other hand, it generates waypoints that fully exploit the omnidirectional capabilities of the robot which leads to trajectories with the shortest initial and optimized travel times. The hybrid planner uses Voronoi paths which results in substantially different waypoints. Nevertheless, the optimization results are comparable, but are achieved in a much shorter planning time. In contrast, the 2D value iteration plans with the circumcircle of the robot. Thus, it has to take detours in some tasks to avoid the narrow passage between P1 and P2 which increases execution times. Additionally, its waypoints have to maintain a larger distance to obstacles in general, which also causes longer initial paths. However, our optimization can compensate this drawback to a large degree by adjusting the trajectory shape, which causes a substantial decrease in execution time as shown in Tab. 1.

In summary, our method is robust to the choice of initial planner and the Voronoi-based planner offers fast computation of initial paths that account for the orientation of the robot. Furthermore, it provides conservative initial paths with respect to obstacle distance that the optimization moves closer to obstacles where beneficial and while there is still planning time left.

Fig. 15 The grid map used for our trajectory optimization experiments, obstacles are shown in black. The map also shows the robot contour at start/goal poses that we use to generate 72 navigation tasks.

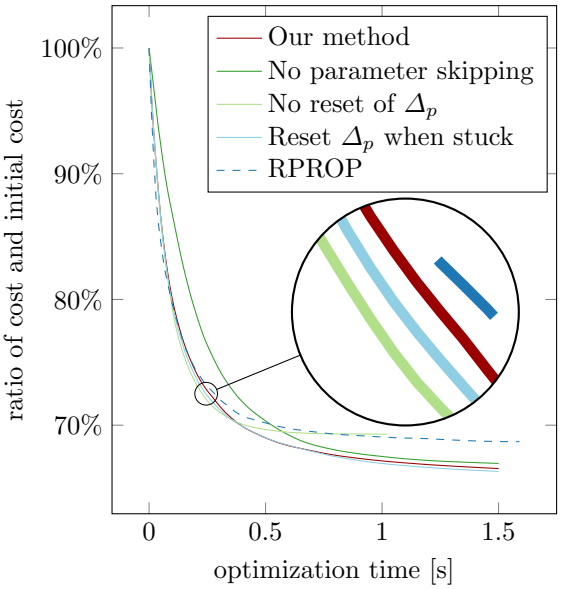

Fig. 16 The progress of trajectory optimization over time, averaged over the 72 navigation tasks defined in Fig. 15. The plot compares different variants of our proposed method from Alg. 1 (see text) and RPROP. To average over different navigation tasks with different potential for optimization, the plot shows the ratio between current trajectory cost and cost of the initial trajectory over time.

7.2 Trajectory Optimization

In this experiment we evaluate the convergence of the trajectory optimization and analyze the effect of different choices in the optimization algorithm. Fig. 15 shows the grid map together with 9 start/goal poses that we used to generate the 72 navigation tasks for this experiment. For each start/goal combination we computed an initial trajectory and optimized it for 1.5 s using the method proposed in Sec. 5.5. To aggregate data over the different tasks, we normalize the trajectory cost during optimization by computing a ratio to the initial cost before optimization. The plot in Fig. 16 shows the average over all 72 tasks for different variants of the optimization algorithm. In general, there are bigger gains in the beginning of the optimization and the convergence rate lowers over time until it reaches a gain of 30% to 35%. The standard deviation lies within 12% for all data points and is not shown in the plot for readability. The high variance is caused by the different potential for optimization in the navigation tasks: for some start/goal combinations the initial trajectory is already close to the optimal solution, e.g., for the tasks that correspond to driving a straight line in the middle of a corridor.

In Fig. 16 we compare the performance of our algorithm against RPROP [37] and against variants of our method in Alg. 1 with different behavior regarding switching to the next parameter upon an improvement (line 19) and resetting the step length for parameters (line 8,22). In contrast to our algorithm, RPROP works on all parameters simultaneously and needs an extra step to compute the partial derivative for each parameter. After a slightly better convergence in the first few steps, it converges at a slightly lower rate to an about 5% worse improvement over the initial trajectories than the proposed method. Since one iteration also takes longer for RPROP than for our algorithm, it creates feasible solutions at a lower frequency (but with bigger improvements between them). This is a disadvantage when trying to completely exploit an allotted time window for optimization, in the plot we see that it overshot the planning time by 0.1 s before creating the next feasible solution. The variant of the proposed method that performs no reset of the parameter step length Δ_p (line 8 and 22 not present) converges to a similar improvement as RPROP, 5% worse than the variants that reset the step lengths. After 0.1 s it shows for a short time a better convergence than RPROP and the variants that reset the step length. Then, this advantage fades and after about 1 s it aborts optimization as no further improvements could be found. The variants that reset the step size in every round (line 8) and only when stuck (line 22) show comparable performance with a slight advantage for the variant that only resets when stuck. As can be seen from the plot, convergence behavior greatly benefits from switching to the next parameter upon improvement (line 19): the variant without this manages to converge to a comparable improvement, but at a slower rate.

In summary, our optimization procedure results in high improvements on the cost function in the first few iterations. Furthermore, we observe a beneficial contribution by continuing with the next parameter as soon as an improvement has been found and by resetting the parameter step size in our optimization algorithm.

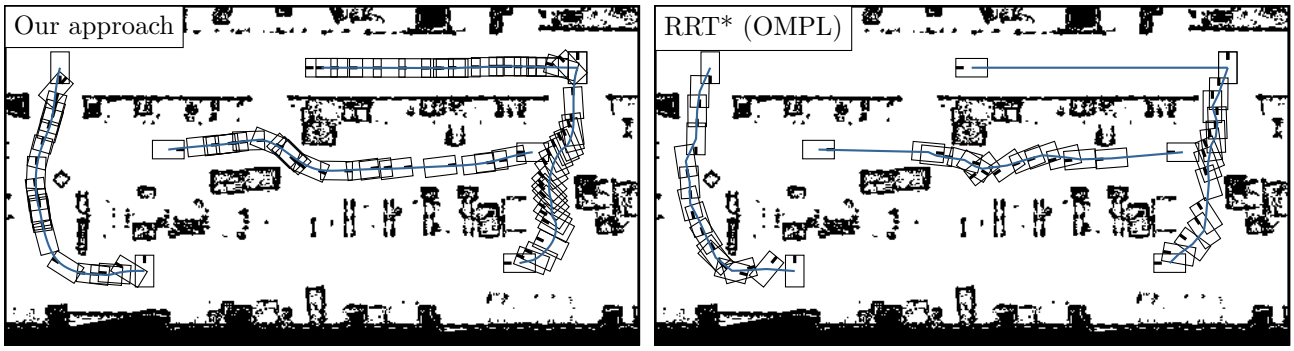

Fig. 17 Comparison of trajectory generation by our approach (left) and a procedure based on the implementation of RRT* [21] in OMPL [45] (right). The figure shows the results for a selection of the 72 tasks defined by the start/goal poses shown in 15. The robot center is shown as blue solid line with an overlay of the robot contour along the trajectory.

We also observe an improvement over direct application of the RPROP algorithm. In addition, our optimization better adheres to the time deadline, which is important to timely deliver updated trajectories for a moving robot. Exceeding the time deadline means that the robot will have passed the starting point of the updated trajectory when it becomes available, resulting in a discontinuous join.

7.3 Comparison of Trajectory Generation

The goal of our proposed method is to provide smooth, curvature continuous and time-optimized navigation that accounts for kinematic and dynamic constraints and limits the robot's speed in the vicinity of obstacles. In this experiment, we compare our method to navigation based on the widely spread motion planning framework Open Motion Planning Library (OMPL) [45].

In contrast to our specialized method, the planners provided in OMPL are very flexible and applicable to high-dimensional search spaces. To generate smooth, curvature continuous trajectories with these planners one needs to resort to sampling in acceleration control space. However, the planners available for these control spaces do not optimize an objective, i.e., they stop after finding a path to the goal. Optimizing planners are available for the geometric domain, for example the RRT* planner [21] that converges to shortest paths consisting of linear connections in the work space, SE(2) in our case. To generate feasible trajectories from the resulting paths, we compute a velocity profile as for our approach (see Sec. 5.4). Due to the curvature discontinuities at the join points of the linear segments, this profile decelerates the robot to a brief stop at these points.

For this experiment we also resort to the 72 navigation tasks given by the start/goal poses in Fig. 15. Both methods were given 1.5 s of total planning time. A selection of the resulting trajectories for both approaches is shown in Fig. 17. As can be seen, the path generated by RRT* is non-smooth at the join points of linear segments and it keeps little distance to obstacles (as RRT* optimizes for path length). In comparison, the trajectories generated by our method are smooth and curvature continuous and keep a safe distance to nearby obstacles.

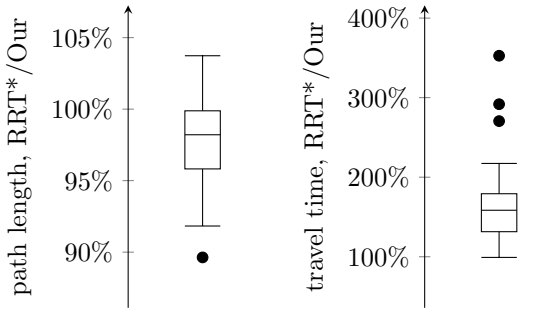

Fig. 18 Comparison of trajectory generation by our approach and a procedure based on the implementation of RRT* [21] in OMPL [45]. The plots show the ratio of path length and travel time of RRT* and our approach for the 72 navigation tasks in Fig. 15. The boxplots show lower and upper quartile along with the median (line). The whiskers indicate data within 1.5 times the interquartile range (IQR) of lower and upper quartile, respectively. Data outside this range is shown as dots. Examples for the generated trajectories are shown in Fig. 17.

A quantitative evaluation of the differences between the two approaches is given by the statistics shown in Fig. 18. The boxplots show the distribution of values for the ratios of path length and travel time between the trajectories generated by RRT* and the proposed method for the 72 navigation tasks. For the most part, the paths computed by RRT* are shorter than the trajectories returned by our method Fig. 18 (left). This is a direct consequence of the different objectives of the two methods: While RRT* optimizes for path length, our method aims to reduce travel time while respecting a set of kinematic and dynamic constraints as well as obstacle imposed speed limits. Together with the stops

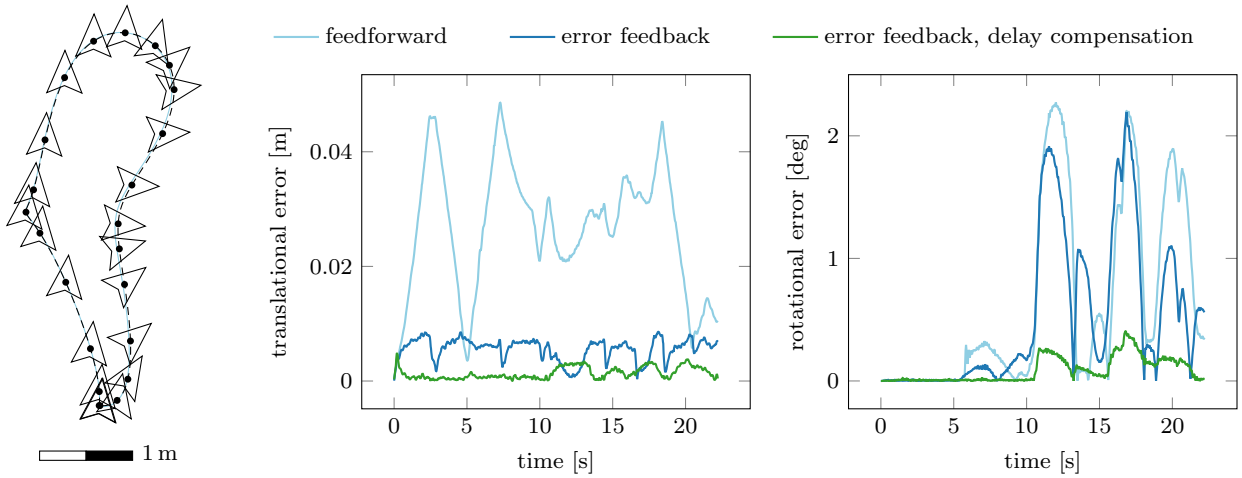

Fig. 19 Execution of a reference trajectory by a real robot. The left part shows the reference trajectory (dashed), indicating the orientation of the robot with darts drawn every second. The left part also shows a feedforward execution (solid). The middle and right plot show the translational tracking error $\|\mathcal{T}(t) - \mathbf{o}(t)\|$ and the rotational tracking error $|\mathcal{T}_\theta(t) - \mathbf{o}_\theta(t)|$ over time, respectively. The plots show the errors for feedforward execution ($\Delta t_{\text{del}} = 0, g_t = 0, g_r = 0$), error feedback ($\Delta t_{\text{del}} = 0, g_t = 3, g_r = 0.3$), and for error feedback with delay compensation ($\Delta t_{\text{del}} = 0.06, g_t = 3, g_r = 0.3$).

at join points of linear segments this explains the higher travel time for the trajectories generated from the RRT* paths: Fig. 18 (right) shows that the proposed method consistently generates trajectories that are 50% faster than the ones returned by computing a velocity profile for the paths generated by RRT*.

In summary, the comparison to the RRT*-based trajectory generation highlights the importance of generating smooth paths with velocity profiles. Our solution accounts for these aspects while optimizing travel time as a cost function. In our system, travel time does not only depend on a number of constraints imposed on the platform and its load but we additionally incorporate the distance to obstacles to achieve safe and intuitive behavior on the shop floor.

7.4 Open-loop and Closed-loop Trajectory Execution

In this experiment, we evaluate the suitability of our trajectory representation on the basis of tracking errors for executing a reference trajectory on a real robot. We perform this experiment with and without error feedback and delay compensation in the controller (see Sec. 6, Eq. (15)). More specifically, we once set $\Delta t_{\text{del}} = 0\,\text{s}$, $g_t = 0$, $g_r = 0$ (feedforward), once $\Delta t_{\text{del}} = 0\,\text{s}$, $g_t = 3$, $g_r = 0.3$ (error feedback) and once $\Delta t_{\text{del}} = 0.06\,\text{s}$, $g_t = 3$, $g_r = 0.3$ (error feedback and delay compensation). For this experiment, we send the reference trajectory to the robot only once and do not update it during execution, the whole experiment takes place with respect to the odometry frame of the robot.

The reference trajectory is shown in Fig. 19 (left) and specifies a loop in an area of $10\,\text{m}^2$ with roughly 22 s of travel time and exploits the holonomic capabilities of our platform. In the figure, the orientation of the robot is denoted by darts, and the driven path for a feedforward execution is shown as solid line. Fig. 19 also shows plots of the translational (middle) and rotational (right) deviation of the actual robot pose from the planned pose for every point in time, as measured by the wheel encoder based odometry of the robot.

When executing the reference trajectory with feedforward commands only, the system relies on the appropriate modeling of the trajectory and its velocity profile. The average errors for this execution are 0.027 m with a standard deviation of 0.01 m and 0.6 degrees with a standard deviation of 0.7 degrees and never exceed 0.05 m and 3 degrees. We attribute these errors to execution inaccuracies and approximation errors introduced by sampling the trajectory with discrete commands sent to the robot. The run with error feedback shows how these errors can be compensated by the controller. The run with delay compensation illustrates this measure's contribution to tracking performance. Accounting for delay with the parameter Δt_{del} reduces the average tracking error from 0.0056 m and 0.52 degrees to 0.0014 m and 0.09 degrees. This effect was even stronger in an earlier version of the omniRob, where we had to account for roughly double the amount of delay in the system.

In summary, the low tracking errors achieved show that the compact path model and velocity profiles used by our approach generate trajectories that can accurately be followed by the robot. We attribute this to the curvature continuity of our paths and the pre-planned

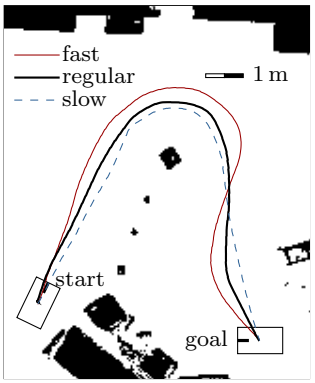

Fig. 20 Setup for an experiment to resemble a medium range transportation task. The figure shows the path of the robot in three exemplary executions with different limits for speed and acceleration in the computation of velocity profiles. With higher admissible speeds the system chooses longer paths that can be traversed faster while staying within obstacle imposed speed limits and constraints on centripetal acceleration.

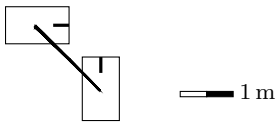

Fig. 21 Experimental setup to resemble a repetitive pick&place task. The figure shows the start/goal poses and an overlay of the 19 trajectories the robot drove in two minutes.

velocities that respect the platform limits. We also observe that our error feedback control scheme with delay compensation is well-suited for trajectory execution. In application scenarios in which a good dynamic model of the platform is available one could expect even better tracking results with model predictive control approaches.

7.5 Influence of Trajectory Constraints

This experiment shows how our system adapts to different values for the constraints in velocity profile generation, see Sec. 5.4. The experimental setup is shown in Fig. 20 and is inspired by a medium range transportation task. The figure shows the driven paths for three exemplary executions of the task with different settings for maximum allowed velocity and acceleration. Lower speed limits lead the system to choose shorter paths that are closer to obstacles. A reduction of the speed limits leads to a reduction of the area in which obstacle speed limits are the dominating constraint and in turn a reduction in path path length leads to shorter travel times. For higher speed limits, the optimization chooses longer paths that feature higher clearance from obstacles and wider curves. The higher possible velocities

Fig. 22 Our experiment for navigation in a dynamic environment. The robot looped for 2.9 km and 2 hours between the eight shown waypoints while object rearrangements and moving people changed the environment. The figure shows some of the driven paths, deviations from the typically chosen route are due to dynamic obstacles blocking the usual path of the robot.

at greater distance from obstacles and in wider curves make up for the increased length of the path.

7.6 Trajectory Execution Accuracy

In these experiments we quantify the performance of our system in executing the planned trajectories. We evaluate the system on the transportation task introduced above (see Fig. 20) as well as on a task that is shown in Fig. 21: Here, the robot has to repeatedly travel between two close-by positions while changing orientation. This task is designed to resemble a repetitive pick&place application for a mobile manipulator. Both tasks have been executed with regular velocity limits and with higher speed limits. We executed the transportation task ten times, yielding an average travel time of 35.4 s with a standard deviation of 0.2 s. Five executions of the variant with raised velocity limits led to a reduced average travel time of 28.4 s again with a standard deviation of 0.2 s. The pick&place task was executed for two minutes yielding 19 very similar trajectories that are shown as overlay in Fig. 21. In a run with higher admissible velocities and accelerations, the robot achieved 30 iterations in the same time interval.

We also evaluated the system in a task that consisted of navigation in a dynamically changing environment. Fig. 22 shows a grid map of the environment together with the 8 waypoints between which the robot traveled repeatedly for 2 hours and 2.9 km. While the robot was traveling, the system had to cope with people blocking its path and changing arrangements of objects in the environment [44].

Fig. 24 The KUKA Moiros robot using our omnidirectional navigation system during a live demonstration at the Hannover Messe 2013 fair.

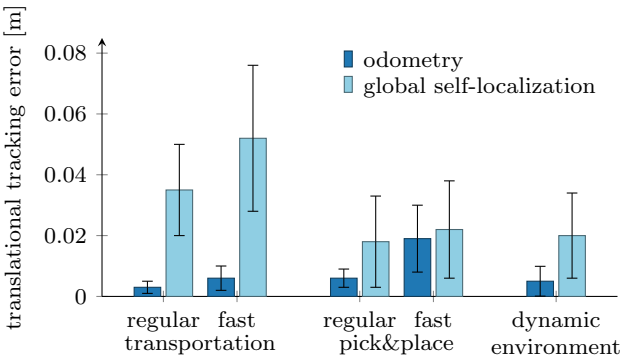

Fig. 23 Mean translational tracking error and standard deviation for the transportation task shown in Fig. 20, the pick&place task shown in Fig. 21, and the navigation in a dynamic environment shown in Fig. 22. The plot shows the error computed in the odometry frame of the robot as well as with respect to global self-localization.

The translational tracking errors for all tasks are shown in Fig. 23. The plot shows the error computed within the odometry frame of the robot as well as with respect to the global self-localization based on a grid map and the two laser range finders of the platform (see Sec. 4). The errors are the deviation of the robot from its planned pose, averaged over time. When we measure the deviation based on the odometry of the robot, the average errors are below 0.02 m. The average errors according to the global self-localization are below the grid map resolution of 0.05 m.

In summary, our approach shows high accuracy when executing the generated trajectories over extended periods of time and in changing environments. The robustness of the overall system builds on the robustness of the employed components for mapping and localization. We attribute the reliability of robot motion generation to the concept of splitting planning and execution of trajectories, see Fig. 2. This allows accurate execution with high frequency error feedback while regular, smooth updating of trajectories accounts for odometry drift, localization noise and changes in the environment. The efficient planning of updated trajectories builds on our compact path model that enables the optimization to adapt the path shape through a small number of parameters. In addition, our procedure for collision checking contributes to the efficiency of trajectory updates.

Fig. 25 The omniRob navigating with the proposed trajectory generation approach in an industrial mobile manipulation demonstration at the KUKA booth at the AUTOMATICA 2010 fair.

7.7 Real-World Deployed Applications

We have extensively tested and deployed the navigation system proposed in this paper. In one application scenario the omniRob had to perform random navigation tasks between three goal locations and never failed to reach the goal in a 3.5 hour trial [38]. The proposed navigation system was also employed in several mobile manipulation demonstrations in the industrial context. The system has been shown on the omniRob at the AUTOMATICA Trade Fair for Automation and Mechatronics in 2010 and 2014 (see Fig. 25) and at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) in 2011. The KUKA Moiros platform, a scaled-up version of the omniRob, used the navigation system in its demonstration at the Hannover Messe 2013, where it won the Robotics Award (see Fig. 24). Additionally, the navigation system has been deployed on the KUKA youBot, a scaled-down version of the omniRob, and demonstrated at the International Conference on Robotics and Automation (ICRA) in 2014.

In addition to the omniRob, the Moiros and the youBot platform, we successfully tested the navigation system on the KUKA omniMove, a large-scale omnidirectional platform and the Willow Garage PR2 robot. Currently, KUKA Laboratories GmbH is shipping a version of the proposed system as prototypical navigation software for the omniRob mobile manipulator.

8 Conclusion

In this paper, we have presented a system for accurate navigation of omnidirectional robots in industrial environments. The system consists of components for

mapping, localization, trajectory generation and robot control. By devising smooth, curvature continuous trajectories we achieve accurate execution with an error feedback controller. Our system optimizes trajectories for travel time meanwhile taking into account platform and safety constraints and leveraging the high maneuverability of omnidirectional robots. Our extensive experiments in simulation and the real world demonstrate the robustness and accuracy of our system and showcase its applicability to the domain of industrial factory floor logistics. This work has a direct impact in the industry as this navigation system is integrated on the industrial mobile omnidirectional platform KUKA omniRob.

References

1. adept mobilerobots. `http://mobilerobots.com`, 2014. Online, accessed 2014-11-26.
2. D. J. Balkcom, P. A. Kavathekar, and M. T. Mason. Time-optimal trajectories for an omni-directional vehicle. *Intl. Journal of Robotics Research (IJRR)*, 25(10):985–999, 2006.
3. J. E. Bobrow, S. Dubowsky, and J. Gibson. Time-optimal control of robotic manipulators along specified paths. *Intl. Journal of Robotics Research (IJRR)*, 4(3):3–17, 1985.
4. O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *The Intl. Journal of Robotics Research*, 21(12):1031–1052, 2002.
5. A. Byravan, B. Boots, S. Srinivasa, and D. Fox. Space-time functional gradient optimization for motion planning. In *IEEE Intl. Conference on Robotics and Automation (ICRA)*, pages 6499–6506, May 2014. doi: 10.1109/ICRA.2014.6907818.
6. Carmen robot navigation toolkit. `http://carmen.sourceforge.net`, 2014. Online, accessed 2014-11-26.
7. J. Connors and G. Elkaim. Manipulating B-Spline based paths for obstacle avoidance in autonomous ground vehicles. In *National Meeting of the Institute of Navigation*, San Diego, USA, 2007.
8. F. Dellaert and M. Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *Intl. Journal of Robotics Research (IJRR)*, 25(12):1181–1203, 2006.
9. F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo Localization for mobile robots. In *IEEE Intl. Conference on Robotics and Automation (ICRA)*, 1999.
10. M. Foskey, M. Garber, M. C. Lin, and D. Manocha. A voronoi-based hybrid motion planner. In *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS)*, 2001.
11. D. Fox. Adapting the sample size in particle filters through KLD-sampling. *Intl. Journal of Robotics Research (IJRR)*, 22(12):985–1003, 2003.
12. D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, Mar. 1997.
13. T. Fraichard and V. Delsart. Navigating dynamic environments with trajectory deformation. *Journal of Computing and Information Technology*, 17, 2009.
14. Frog AGV Systems. `http://frog.nl`, 2014. Online, accessed 2014-11-26.
15. G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard. A tutorial on graph-based SLAM. *Intelligent Transportation Systems Magazine, IEEE*, 2(4):31–43, 2010.
16. E. Guizzo. Three engineers, hundreds of robots, one warehouse. *Spectrum, IEEE*, 45(7):26–34, 2008.
17. J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line-simplification algorithm. Technical report, University of British Columbia, 1992.
18. A. Hornung, M. Phillips, E. G. Jones, M. Bennewitz, M. Likhachev, and S. Chitta. Navigation in three-dimensional cluttered environments for mobile manipulation. In *IEEE Intl. Conference on Robotics and Automation (ICRA)*, 2012.
19. M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE Intl. Conference on Robotics and Automation (ICRA)*, pages 4569–4574, May 2011. doi: 10.1109/ICRA.2011.5980280.
20. T. Kalmár-Nagy, R. D'Andrea, and P. Ganguly. Near-optimal dynamic trajectory generation and control of an omnidirectional vehicle. *Robotics and Autonomous Systems*, 46(1):47–64, 2004.
21. S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Intl. Journal of Robotics Research (IJRR)*, 30(7):846–894, 2011.
22. R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *IEEE Intl. Conference on Robotics and Automation (ICRA)*, 2011.
23. R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. Autonomous robot navigation in highly populated pedestrian zones. *Journal of Field Robotics*, 2014.
24. F. Lamiraux, D. Bonnafous, and O. Lefebvre. Reactive path deformation for nonholonomic mobile robots. *IEEE Transactions on Robotics*, 20(6):967–977, 2004.

25. B. Lau, C. Sprunk, and W. Burgard. Kinodynamic motion planning for mobile robots using splines. In *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS)*, 2009.
26. B. Lau, C. Sprunk, and W. Burgard. Efficient grid-based spatial representations for robot navigation in dynamic environments. *Robotics and Autonomous Systems*, 61(10):1116–1130, 2013.
27. M. Likhachev and D. Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *Intl. Journal of Robotics Research (IJRR)*, 28 (8):933–945, 2009.
28. Y. Liu, J. J. Zhu, R. L. Williams II, and J. Wu. Omni-directional mobile robot controller based on trajectory linearization. *Robotics and Autonomous Systems*, 56(5), 2008.
29. E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige. The office marathon: Robust navigation in an indoor office environment. In *IEEE Intl. Conference on Robotics and Automation (ICRA)*, 2010.
30. K. Maček, G. Vasquez, T. Fraichard, and R. Siegwart. Towards safe vehicle navigation in dynamic urban scenarios. *Automatika*, 50(3-4):184–194, 2009.
31. D. Montemerlo, N. Roy, and S. Thrun. Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. In *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS)*, 2003.
32. P. Muir. *Modeling and Control of Wheeled Mobile Robots*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1988.
33. E. Olson. *Robust and efficient robotic mapping*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2008.
34. O. Purwin and R. D'Andrea. Trajectory generation and control for four wheeled omnidirectional vehicles. *Robotics and Autonomous Systems*, 54:13–22, 2006.
35. S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *IEEE Intl. Conference on Robotics and Automation (ICRA)*, pages 802–807, 1993.
36. N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE Intl. Conference on Robotics and Automation (ICRA)*, pages 489–494, 2009.
37. M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Intl. Conf. on Neural Networks*, 1993.
38. J. Roewekaemper, C. Sprunk, G. D. Tipaldi, C. Stachniss, P. Pfaff, and W. Burgard. On the position accuracy of mobile robot localization based on particle filters combined with scan matching. In *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS)*, 2012.
39. R. Rojas and A. G. Förster. Holonomic control of a robot with an omnidirectional drive. *Künstliche Intelligenz*, 20(2):12–17, 2006.
40. M. Rufli, D. Ferguson, and R. Siegwart. Smooth path planning in constrained environments. In *IEEE Intl. Conference on Robotics and Automation (ICRA)*, 2009.
41. J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: Science and Systems*, volume 9, pages 1–10, 2013.
42. K. G. Shin and N. D. McKay. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Transactions on Automatic Control*, 30(6):531–541, 1985.
43. C. Sprunk, B. Lau, P. Pfaff, and W. Burgard. Online generation of kinodynamic trajectories for non-circular omnidirectional robots. In *IEEE Intl. Conference on Robotics and Automation (ICRA)*, 2011.
44. C. Sprunk, J. Roewekaemper, G. Parent, L. Spinello, G. Tipaldi, W. Burgard, and M. Jalobeanu. An experimental protocol for benchmarking robotic indoor navigation. In *Proc. of the International Symposium on Experimental Robotics (ISER)*, 2014.
45. I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, Dec. 2012. `http://ompl.kavrakilab.org`.
46. swisslog. `http://swisslog.com`, 2014. Online, accessed 2014-11-26.
47. S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial intelligence*, 128(1):99–141, 2001.
48. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.
49. 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*, 23(9):661–692, 2006.
50. G. D. Tipaldi, L. Spinello, and W. Burgard. Geometrical flirt phrases for large scale place recognition in 2d range data. In *IEEE Intl. Conference on Robotics and Automation (ICRA)*, 2013.
51. G. D. Tipaldi, M. Braun, and K. O. Arras. Flirt: Interest regions for 2d range data with applications

to robot navigation. In *Experimental Robotics*, pages 695–710. Springer, 2014.
52. N. Tomatis. Bluebotics: Navigation for the clever robot [Entrepreneur]. *Robotics Automation Magazine, IEEE*, 18(2):14–16, June 2011.
53. K. Watanabe. Control of an omnidirectional mobile robot. In *Proc. of Intl. Conference on Knowledge-Based Intelligent Electronic Systems*, 1998.
54. M. Werling and L. Gröll. Low-level controllers realizing high-level decisions in an autonomous vehicle. In *IEEE Intelligent Vehicles Symposium*, 2008.
55. P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9, 2008.
56. Y. Yang and O. Brock. Elastic roadmaps–motion generation for autonomous mobile manipulation. *Autonomous Robots*, 28(1):113–130, 2010. ISSN 0929-5593.
57. J. Ziegler and C. Stiller. Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios. In *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS)*, 2009.
58. J. Ziegler, M. Werling, and J. Schröder. Navigating car-like robots in unstructured environments using an obstacle sensitive cost function. In *IEEE Intelligent Vehicles Symposium (IV 08)*, 2008.