Kinodynamic Motion Planning for Mobile Robots Using Splines

Boris Lau Christoph Sprunk Wolfram Burgard

***Abstract*— This paper presents an approach to time-optimal kinodynamic motion planning for a mobile robot. A global path planner is used to generate collision-free straight-line paths from the robot's position to a given goal location. With waypoints of this path, an initial trajectory is generated which defines the planned position of the robot over time. A velocity profile is computed that accounts for constraints on the velocity and acceleration of the robot. The trajectory is refined to minimize the time needed for traversal by an any-time optimization algorithm. An error-feedback controller generates motor commands to execute the planned trajectory. Quintic Bézier splines are used to allow for curvature-continuous joins of trajectory segments, which enables the system to replan trajectories in order to react to unmapped obstacles. Experiments on real robots are presented that show our system's capabilities of smooth, precise, and predictive motion.**

I. INTRODUCTION

Motion planning is a fundamental task for wheeled mobile robots. It consists of planning a path from the robot's position to a given goal location using a representation of the environment, and computing motion commands that make the robot platform follow this path [1]. Most traditional motion planning systems use a global path planner like A* or its descendants, e.g., [2], [3], [4], which find the shortest path on a 2D grid or graph that represents the traversable space. These paths typically contain sharp corners and can only be accurately followed by stopping and turning on the spot, which significantly increases the time of travel.

The generation of actual motor commands is therefore often carried out by reactive systems [5], [6], [7]. These consider the vector to the next one or two waypoints in the planned path and the distance to obstacles perceived by the robot's sensors. The Dynamic Window [8] additionally considers the platform's kinodynamic constraints. All of these approaches have in common that the robot deliberately deviates from the planned path to drive smooth curves, which leads to faster progress towards the goal location.

The downside of this solution is, that (a) optimality properties of the straight line path do not apply to the resulting continuous trajectory and no time of travel optimality is achieved, (b) the shape of the path, e.g., how much corners are cut, depends on parameter settings rather than optimization or search, (c) velocities and accelerations are not planned in advance but subject to reactive behavior, which prevents accurate motion prediction and makes satisfaction of hard constraints difficult, and (d) no guarantees can be made for the control stability or convergence behavior of the system.

All authors are with the University of Freiburg, Germany, Department of Computer Science {lau,sprunkc,burgard}@informatik.uni-freiburg.de.

This work was partly funded by the European Commission under contract number FP6-IST-045388.

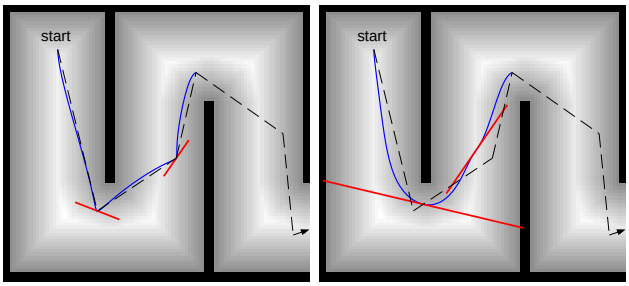

Fig. 1. Trajectory before (left) and after (right) optimization, generated with sparse waypoints taken from a straight-line path (dashed). Tangents at inner waypoints are depicted by red lines. Trajectories are shown in blue on the distance map, where darker values are closer to obstacles.

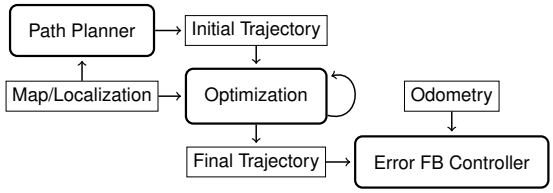

Fig. 2. System overview. A global path planner generates sparse waypoints, from which an initial trajectory is created. The optimization reduces the time needed for traversal. It checks the trajectories for collisions using a map and self-localization. An error-feedback controller executes the final trajectory. It uses odometry data to assess its deviation from the path.

Robots like autonomous cars, wheelchairs, autonomous transport vehicles, and other service robots can carry heavy or sensitive payload, necessitate precise motion, or might be required to show predictive behavior and therefore demand solutions to the above-mentioned problems.

This paper presents an approach to time-optimal kinodynamic motion planning. It generates a parametric curve from sparse waypoints that is augmented with a velocity profile to control the robot's position in space over time. This curve is optimized to yield a smooth and time-optimal trajectory that complies with kinodynamic constraints of the robot, as shown in Fig. 1. An error-feedback controller is used to steer the robot along the trajectory. In this way, our approach addresses all the problems mentioned above. A schematic overview of the system is given in Fig. 2.

The remainder of the paper is organized as follows. After a discussion of related work, Sect. III introduces quintic Bézier splines and the employed velocity profiles. Then, Sect. IV describes the generation and optimization of the initial trajectory. In Sect. VI we present experimental results, and finally conclude the paper in Sect. VII.

II. RELATED WORK

The problem of *time-optimal kinodynamic motion planning* was first posed by Donald *et al.* as to determine the

motor commands that guide a robot from start to goal on a collision-free path in minimal time while respecting kinodynamic constraints [9]. Stachniss and Burgard included discretized translational and rotational velocities into a five-dimensional A* search, which returns a series of velocity commands to be issued in discrete time intervals [10]. Deviations from the trajectory are handled by frequently executing the global path planner, which generates a new trajectory for the current position.

Planned trajectories can be precisely tracked using an error-feedback controller, which runs in a control loop that does not alter the trajectory. As input, a trajectory representation is desired that continuously defines position, velocities, and accelerations over time. To this end, some authors have used cubic Bézier splines [11], [12]. Howard and Kelly proposed spline interpolation of velocity commands to generate such trajectories [13]. Likhachev and Ferguson perform a search on a discrete set of these action primitives to obtain a smooth trajectory for an autonomous car [14]. So far, all of these approaches have in common that the generated curves represent continuous position changes, but are not continuous in curvature, which induces abrupt rotational accelerations. In [14] this problem is acknowledged and alleviated by limiting the car's velocity to 2 m/s at high curvatures.

Gulati and Kuipers emphasize the need for smooth and comfortable motion when transporting people, and propose the use of cubic B-Splines [15], as also done by Shiller and Gwo [16]. While cubic B-splines can indeed be curvature continuous from start to end, local changes affect the shape of up to six neighboring segments, and therefore it is not practically feasible to attach or replace a segment of an existing trajectory with curvature continuity at an arbitrary point of the spline. To achieve this, we propose to use *quintic Bézier splines*, which are curvature continuous and additionally allow for attaching segments without curvature discontinuities. In this way, trajectories can be changed on the fly in order to react to sudden changes, for example in the presence of dynamic obstacles.

In contrast to approaches that perform a search on a discretized action set like [14], [17], [18], Connors and Elkaim propose to perform optimization of parameterized trajectories [19]. Their system starts with straight-line segments and iteratively moves control points, aiming for a smooth and collision-free trajectory, which is however not guaranteed to be achievable. Our approach also employs optimization to obtain fast and smooth motion, but starts from an initial straight-line path that is collision-free. Thus, it has any-time characteristics, i.e., it can be interrupted at any point in time to retrieve a collision-free solution.

Most of the existing approaches determine just the shape of the planned trajectory by search or optimization, but rely on local heuristics to set the translational velocities [12], [14], [15], [18], [19]. However, in order to determine truly time-optimal trajectories, the admissible velocities governed by constraints on velocities and accelerations have to be considered. In the work by Macek *et al.* [17] and Stachniss and Burgard [10], the velocity is part of the search, but the systems are limited to a discretized action space. Shiller and Gwo [16] consider a curve that defines the shape of the trajectory, and generate a corresponding velocity profile that considers constraints on the translational velocity and acceleration for an Ackermann drive.

Our system is similar to this approach, but adds the capability of curvature continuous replanning. During optimization, it considers constraints on translational and rotational velocities and accelerations. Furthermore, the maximum allowed centripetal acceleration and a speed limit in the vicinity of obstacles are considered. In this way it generates space-time trajectories that are time-optimal and traversable.

III. REPRESENTATION OF TRAJECTORIES

We consider the trajectory $\langle x, y\rangle = \hat{Q}(t)$ of a robot as its position on the ground in world coordinates over time. Similar to previous approaches [13], [16], we represent the shape and the planned translational velocity of the robot independently. The shape of the trajectory is defined by a parametric curve $Q(u)$. Its non-linear internal parameter u is mapped to the metric distance s along the path via numerical arc-length parameterization, $s = f_s(u)$. The translational velocity of the robot as a function of s is represented by a velocity profile $v(s)$. Integrating the velocity profile creates a mapping from distance s to time, $t = f_t(s) = \int_0^s 1/v(S)dS$.

Chaining and inverting these mappings finally yields the trajectory, $\hat{Q}(t) = Q(f_s^{-1}(f_t^{-1}(t)))$.

A. Quintic Bézier splines as parametric curves

The curve $Q(u)$ defines the 2D shape of the trajectory. It consists of N_Q joined segments $Q_i(u_i)$, $i \in [0, N_Q\text{–}1]$. While $u_i \in [0, 1]$ parameterizes the i-th segment from its start to end, the variable $u \in [0, N_Q]$ is the internal parameter of the whole curve and results from chaining the u_i. Thus, the trajectory can be denoted by $Q(u) = Q_{\lfloor u \rfloor}(u - \lfloor u \rfloor)$, where $\lfloor u \rfloor$ is the truncation of u to the next smaller integer.

To achieve smooth motion, the curvature $c(u)$ of the curve $Q(u)$ is required to be continuous. It is given by

$$c(u) = \left(Q'_x(u)Q''_y(u) - Q'_y(u)Q''_x(u)\right) / \|Q'(u)\|^3 \,. \quad (1)$$

We obtain continuous curvature over the entire curve by joining the segments with C^2 continuity, i.e., the curve itself, its first derivative $Q'(u)$ and second derivative $Q''(u)$ are continuous. To this end, the starting waypoint W_{i+1} of segment Q_{i+1} is also the end point of the previous segment i, thus $W_{i+1} = Q_{i+1}(0) = Q_i(1)$. The same holds for the starting tangent (first derivative) $T_{i+1} = Q'_{i+1}(0) = Q'_i(1)$, and the second derivative $A_{i+1} = Q''_{i+1}(0) = Q''_i(1)$, for $i \in \{0, \ldots, N_Q - 1\}$. The widely used cubic B-splines fulfill the property of *(a)* C^2 continuity and *(b)* locality, i.e., moving a control point only affects a few neighboring segments. However, they do not have another property required for effective motion planning at the same time, namely *(c)* control point interpolation, i.e., the curve passes through the start and end control point of a segment. This constraint is met by cubic Bézier curves and Catmull-Rom splines, but they cannot be C^2 continuous and local at the same time.

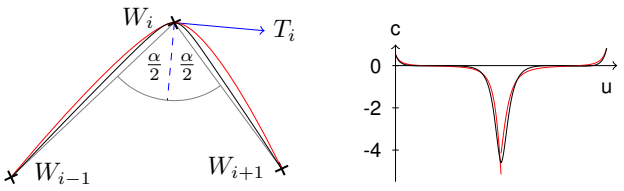

Fig. 3. *Left:* Two splines, one is composed of two *cubic* Bézier spline segments (red) that are joined with C^1 continuity at W_i, and the other of two *quintic* Bézier spline segments (black), that are joined with C^2 continuity at W_i using a heuristic for the second derivatives (see text). The tangent T_i (blue) is orthogonal to the bisector of angle α (dashed blue) in both cases. *Right:* Curvature c of the curves shown left. The cubic spline (red) has discontinuous curvature, while the quintic spline (black) does not.

To smoothly join two curve segments, e.g., when modifying an existing trajectory to react to sudden changes, we furthermore require *(d)* the possibility of locally choosing the second derivative at the ends of a curve. With polynomials, all of the properties (a)–(d) can be fulfilled simultaneously only for degree five and higher. Non-polynomial curve types, like Clothoid splines, are much more complicated to position in space, and to our knowledge no solution exists to smoothly join two Clothoid splines at arbitrary points of the curve. Therefore, we choose quintic Bézier splines as means of curve representation, which are derived from fifth-degree Bernstein polynomials:

$$Q_i(u_i) = (1-u_i)^5 P_{i,0} + 5(1-u_i)^4 u_i P_{i,1} + 10(1-u_i)^3 u_i^2 P_{i,2} + 10(1-u_i)^2 u_i^3 P_{i,3} + 5(1-u_i) u_i^4 P_{i,4} + u_i^5 P_{i,5},$$

where $P_{i,0}$ to $P_{i,5}$ are the six points that control the curve's shape. By computing $Q(u_i)$, $Q_i'(u_i)$ and $Q_i''(u_i)$ at the start ($u_i = 0$) and end ($u_i = 1$) of a segment one can see that $P_{i,0}$ and $P_{i,5}$ are the start/end points of the i-th segment, and thus $P_{i,0} := W_i$, and $P_{i,5} := W_{i+1}$. Furthermore, $P_{i,1}$ and $P_{i,4}$ are determined by the tangents T_i and T_{i+1} at the start and end of the segment, and equivalently, $P_{i,2}$ and $P_{i,3}$ are set using the second derivatives A_i and A_{i+1}:

$$P_{i,1} := W_i + \tfrac{1}{5} T_i, \qquad P_{i,4} := W_{i+1} - \tfrac{1}{5} T_{i+1},$$
$$P_{i,2} := \tfrac{1}{20} A_i + 2P_{i,1} - W_i, \qquad P_{i,3} := \tfrac{1}{20} A_{i+1} + 2P_{i,4} - W_{i+1}$$

The orientation of T_0, the start tangent of the trajectory, is equal to the orientation θ of the robot at the start position W_0. The other tangents depend on the straight lines connecting the waypoints W_i. The tangent at the end of the curve points in the direction of the last line segment. The orientation of the tangents at the inner waypoints is determined using a heuristic commonly used for cubic Bézier curves [20], which sets the angle of the tangent T_i at a waypoint W_i perpendicular to the angular bisector of the neighboring line segments, as shown in Fig. 3 (left). The magnitude $\|T_i\|$ of the tangent is set to half the Euclidean distance to the closest neighboring waypoint, multiplied with a scalar elongation factor e_i, which will be varied during optimization, see Sect. IV-B.

We aim to set the *second derivatives* A_i at a waypoint W_i to values that generate smooth curves. Cubic splines minimize the integral of the second derivative's absolute value [21], which in general corresponds to small changes in curvature. This motivates our heuristic, which tries to mimic the behavior of cubic splines while meeting the additional requirements mentioned above. Given the waypoints and tangents, cubic Bézier splines are readily defined for both neighboring segments, and we use a weighted mean of their second derivatives at the join point as the second derivatives of the quintic Bézier spline. The weight is anti-proportional to the normalized length of the joining line segments. Thus, the shape of the joined quintic Bézier spline is similar to the joined cubic Bézier spline, but does not have the curvature discontinuity at the join point W_i.

During optimization, the position of the waypoints W_i and the elongation e_i of the tangents are adjusted. The tangents' orientation and the second derivatives are determined by the above heuristics.

B. Velocity profile with piecewise constant accelerations

The velocity profile $v(s)$ is a piecewise linear function, which relates to piecewise constant translational accelerations. Its support points $\hat{s}_k$ are closely spaced at distances $< 1\,cm$. Each $v_k = v(\hat{s}_k)$ is set to the maximum velocity that is both allowed and feasible, given a curve $Q(u)$, and a set of kinodynamic constraints. As shown in Fig. 4, the profile is computed in three phases to respect these constraints.

1) Isolated constraints: In the first phase (Fig. 4 left), the maximum admissible velocity v_k is computed for each support $\hat{s}_k$ independently from the other v_j, $j \neq k$. It is determined as the maximum velocity that satisfies all imposed constraints described in the following. Without loss of generality, we assume the robot to perform forward motion, i.e., $v_k \geq 0$. Let $v_{\max}$ be the (constant) *maximum translational velocity* of the robot, then $v_k \leq v_{\max}$ is required. If the *maximum rotational velocity* $\omega_{\max}$ is limited, then $v_k \leq \omega_{\max}/|c_k|$ has to hold, with c_k being the trajectory's curvature at point $\hat{s}_k$. Note that for robots equipped with a *differential drive* or *Mecanum wheels*, the limit can also be set as a function of curvature and maximum wheel velocity.

To ensure safe motion in the presence of obstacles, the sum of braking distance and traveled distance during the reaction time t_{react} of the system has to be smaller than the *distance d_{obst} to the closest obstacle*, which leads to the constraint

$$v_k \leq -a_{\text{brake}} t_{\text{react}} + \sqrt{a_{\text{brake}}^2 t_{\text{react}}^2 + 2 a_{\text{brake}} d_{\text{obst}}}\,, \qquad (2)$$

where a_{brake} is the maximum deceleration when braking.

Note that this is conservative, a more complex formulation could account for the robot's direction of motion with respect to the obstacle. Furthermore, to avoid skidding of the vehicle or load damage in curves, the *maximum centripetal acceleration* a_{cent} can be limited with $v_k \leq \sqrt{a_{\text{cent}}/|c_k|}$.

2) Acceleration constraints: In addition to the isolated constraints described above, v_k is also limited by the constraints on v_{k-1} and v_{k+1} in connection with the *maximum translational acceleration* a_{trans} and the *maximum rotational acceleration* a_{rot} of the robot. Furthermore, the start velocity v_0 is given and fixed, e.g., $v_0 = 0$ if the trajectory is planned for the initially standing robot, and possibly the terminal

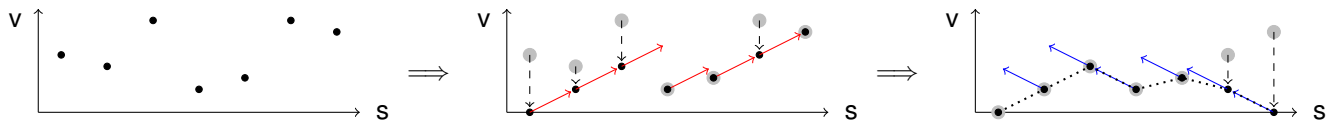

Fig. 4. Velocity profile generation in three phases. Black points denote the admissible velocities v at the supports points. Bigger gray dots mark values inherited from the previous phase. *Left:* The maximum translational velocity v that meets all *isolated constraints* is computed independently for each support. *Middle:* In the second phase, the profile is made consistent for increasing s, given a start velocity and acceleration constraints (red arrows). *Right:* The third phase adds consistency for decreasing s for a given end velocity and deceleration constraints (blue arrows), and yields the final profile (dashed).

velocity as well. In the second phase with increasing k and the third phase with decreasing k (middle and right in Fig. 4), the velocity v_k is set to the maximum value that fulfills the constraints imposed by a_{trans} and a_{rot},

$$v_{k-1} - a_{\text{trans}}\Delta_t \leq \quad v_k \quad \leq v_{k-1} + a_{\text{trans}}\Delta_t, \tag{3}$$

$$v_{k-1}c_{k-1} - a_{\text{rot}}\Delta_t \leq \; v_k c_k \; \leq v_{k-1}c_{k-1} + a_{\text{rot}}\Delta_t, \tag{4}$$

where Δ_t is the travel time needed for the distance between the supports $\hat{s}_{k-1}$ and $\hat{s}_k$. If the acceleration and deceleration constraints are not the same, different values have to be used for a_{trans} in these two phases, a_{accel} and a_{brake}, respectively.

Only for synchro-drive robots, a_{trans} and a_{rot} are totally independent. However, since the mass m of platform and payload governs these limits via translational inertia and rotational moment of inertia, it is reasonable to assume a_{trans} and a_{rot} to be independent for other platforms as well.

The upper bound for the v_k that satisfy both of the above constraints can be determined in closed form. However, the mathematical derivations are quite elaborate since it is a complex function of velocity, acceleration, curvature and curvature change. For the sake of brevity we would like to refer the reader to the work by Sprunk [22] for details.

After the third phase, the velocity profile realizes piecewise constant accelerations, meets all described velocity and acceleration constraints, and is therefore traversable by the robot. With monotonous curvature changes between the closely spaced support points, the constraints hold at the support points and in the intervals between them as well. If the velocity profile was defined over time instead of distances, changing velocities would imply changes of the support points, which clearly complicates the process.

IV. MOTION PLANNING AND OPTIMIZATION

This section describes the generation and optimization of a trajectory. In short, an initial trajectory is generated from sparse waypoints given by a global path planner. The trajectory is refined to minimize the traveling time until a) the optimum is reached or b) planning time is up.

A. Global planning and trajectory generation

We use a global path planner [2] that operates on a high-resolution 2D grid map. To account for unmapped obstacles, the map is regularly updated with data acquired with a laser range finder. The planner returns the shortest traversable path from the robot's position to a given goal location as a straight-line trajectory, where each segment connects two neighboring grid cells in the grid map. This trajectory is collision-free, but contains sharp discontinuous corners. It is pruned to contain only sparse waypoints by replacing

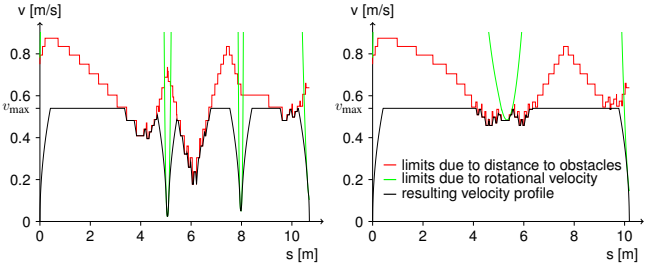

Fig. 5. Velocity profiles for the trajectories in Fig. 1 before (left) and after (right) optimization. The translational velocity $v(s)$ is a function of distance s along the trajectory. It is limited by several *isolated constraints*: a general speed limit ($v_{\max}$), the maximum rotational velocity (green), closeness to obstacles (red), and a limited centripetal acceleration (similar to the green curve, omitted for readability). The constraints on translational and rotational acceleration control the slopes in the resulting profile.

two adjacent line segments with a direct straight line, if the resulting line trajectory is still free of collisions and the resulting segment is shorter than a maximum length.

Because of the sparsity of the waypoints after pruning, the line trajectory segments after the first four waypoints (including the start) are usually out of the robot's field of view in most cases, cf. Fig. 1. In order not to spend computational time on motion planning for areas where unmapped obstacles cannot be perceived and correctly accounted for, we only use the first four waypoints for trajectory generation.

Between each pair of consecutive waypoints, a quintic Bézier spline is created. The initial tangents and the second derivatives are determined by heuristics (see Sect. III-A). To mimic the straight-line trajectory given by the waypoints, a small tangent elongation factor $e = 0.5$ is used (see Sect. III-A), which causes sharp turns at the waypoints. The result is an initial trajectory consisting of three continuously joined spline segments, that can only be traversed at very low speeds, but is collision-free on the other hand, as shown in Fig. 1 (left). This trajectory is refined during the optimization process to minimize the estimated time of travel.

B. Trajectory refinement by optimization

The optimization system takes an initial trajectory (generated as described above) and a distance-transformed map as input. The map contains obstacles that are known a-priori, as well as previously unmapped obstacles that are detected by the robot's sensors.

The optimization changes the shape of the trajectory to reduce a given cost measure by adjusting a set $\mathcal{P}$ of tunable parameters. In our experiments, we use the overall time of travel as the only cost measure, but other costs like the estimated energy consumption could be considered as well. The parameter set consists of the tangent elongation factor e

Algorithm 1 Trajectory optimization based on RPROP.

$\hat{Q}_{\text{best}} \leftarrow$ initial trajectory
$\mathcal{P} \leftarrow$ parameters of initial trajectory
repeat
  $\Delta_{\text{term}} \leftarrow 0$
  for all $p \in \mathcal{P}$ **do**
    $\hat{Q}_{\text{curr}} \leftarrow \hat{Q}_{\text{best}}$
    $d_p \leftarrow d_p^0$
    repeat
      $\hat{Q}_{\text{mod}} \leftarrow \text{MOD}(\hat{Q}_{\text{curr}}, p \leftarrow p + d_p)$
      if $\text{cost}(\hat{Q}_{\text{mod}}) < \text{cost}(\hat{Q}_{\text{best}})$ **then**
        $\Delta_{\text{term}} \leftarrow \max(\Delta_{\text{term}}, \text{cost}(\hat{Q}_{\text{best}}) - \text{cost}(\hat{Q}_{\text{mod}}))$
        $\hat{Q}_{\text{best}} \leftarrow \hat{Q}_{\text{mod}}$
        break
      end if
      $d_p \leftarrow \begin{cases} d_p \cdot 1.2 & \text{cost}(\hat{Q}_{\text{mod}}) < \text{cost}(\hat{Q}_{\text{curr}}), \\ -d_p \cdot 0.5 & \text{else.} \end{cases}$
      $\Delta_{\text{cost}} \leftarrow |\text{cost}(\hat{Q}_{\text{mod}}) - \text{cost}(\hat{Q}_{\text{curr}})|$
      $\hat{Q}_{\text{curr}} \leftarrow \hat{Q}_{\text{mod}}$
    until $\Delta_{\text{cost}} < \epsilon_p$ OR time is up
  end for
until $\Delta_{\text{term}} < \epsilon_{\text{term}}$ OR time is up
return $\hat{Q}_{\text{best}}$

for the first and each of the inner waypoints, as well as the position W_i of the inner waypoints (start and end are fixed).

Fig. 5 depicts the velocity profiles for the trajectories shown in Fig. 1. Optimization alters the shape of the trajectory in a way that alleviates the influence of the constraints on the admissible velocity: the optimized trajectory is faster due to less sharp curves and larger distances from the wall (elongated tangents), and shorter as well (moved waypoints).

The *cost function*, i.e., the total time of travel, depends on the trajectory shape together with the velocity profile, which depends on several constraints and the obstacle distance map. This complex function is not differentiable, therefore we use an optimization scheme inspired by the derivative-free RPROP algorithm by Riedmiller and Braun [23].

The *optimization algorithm*, shown in Algorithm 1, refines a trajectory as follows: in every iteration, as long as planning time is left and the optimization of any of the parameters in the last iteration has brought an improvement bigger than a threshold ϵ_{term}, the parameters are optimized successively: for a chosen parameter $p \in \mathcal{P}$ a new spline is computed by adding an initial offset $d_p := d_p^0$ to that parameter.

Now a *velocity profile* is generated for the modified spline, as described in Sect. III-B. The spline and the velocity profile together form the modified trajectory $\hat{Q}_{\text{mod}}$. From this, the cost, i.e., the time needed to traverse the trajectory is determined. If the trajectory is not collision-free according to the obstacle map, its cost is set to infinity.

If this cost is lower than the cost of the current best trajectory $\hat{Q}_{\text{best}}$, the changed parameter $p := p + d_p$ is kept, and the next parameter is optimized. Otherwise, the parameter p is further optimized with adaptive step sizes as in RPROP: the offset d_p is increased to $1.2d_p$ if the cost is reduced, and inverted/reduced to $-0.5d_p$ if increased. The optimization for that parameter terminates if a modified trajectory is better than the best one, the resulting changes in cost are smaller than a threshold ϵ_p or planning time is up.

The trajectories generated by our approach are time-optimal in the mentioned search space, i.e., parameterized smooth trajectories that are derived from the pruned shortest 2D path. To also consider trajectories that follow a different route, the algorithm can be run in parallel with different straight-line path inputs, e.g., the n-best paths that are topologically different with respect to obstacles.

V. PLAN EXECUTION AND REPLANNING

Assume a planned trajectory $\hat{Q}(t)$, which defines the targeted robot position at time step t. We use the *dynamic feedback linearization controller* developed by Oriolo *et al.* [24] to steer the robot along the trajectory. The second derivative $\hat{Q}''(t)$ of the trajectory is the targeted vector of accelerations in Cartesian coordinates, which the controller uses as feedforward control signal. The deviation of the robot's actual position and velocity from the targeted position $\hat{Q}(t)$ and velocity $\hat{Q}'(t)$ are used as error-feedback signals. The controller combines these signals, and computes the appropriate translational and rotational velocities v and ω that are sent to the robot platform. The reaction time of the platform can be compensated for by adding an offset to t when determining the feedforward input.

As mentioned in Sect. IV-A, the trajectory $\hat{Q}(t)$ only accounts for the first four waypoints and needs to be extended regularly. Periodic replanning is also required to prevent accumulation of odometry errors and to react to changes in the environment. Therefore, the following procedure is executed repeatedly: waypoints of the straight-line path from the predicted robot position $\hat{Q}(t+t_{\text{plan}})$ to the goal location are computed by the global planner, where t_{plan} is the scheduled time span for planning and optimization. A new initial trajectory is generated from these waypoints, that continuously joins $\hat{Q}$ at $t+t_{\text{plan}}$, i.e., the splines join with continuous curvature, and the velocity profiles are continuous as well. This trajectory is optimized until $t+t_{\text{plan}}$ is reached, then the controller switches seamlessly to the new trajectory.

VI. EXPERIMENTS

To test the precision and predictability of motion trajectories planned by our system, we conduct several experiments using a real robot. Additional properties, namely the appropriateness of the optimization parameter set, the resulting shape of the optimization manifold, and the reaction to localization errors are tested using a simulator.

A. Analyzing the optimization manifold

Just as RPROP, our optimization algorithm is able to deal with local minima to some extent. However, an ill-natured optimization manifold could cause suboptimal results. To assess this problem, we have created seven artificially complex environments, and compared trajectories optimized by our system with the best trajectories found by exhaustive

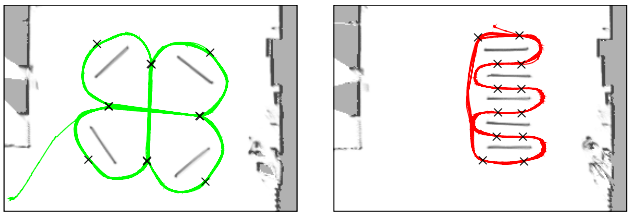

Fig. 6. Trajectory of a Pioneer P3-DX robot overlayed on a grid map for the obstacle courses "clover" (left) and "zigzag" (right). The crosses mark intermediate goal locations. The robot drove 10 rounds in the clover course, and 5 rounds in the zigzag course. The frames cover an area of 14m x 10m.

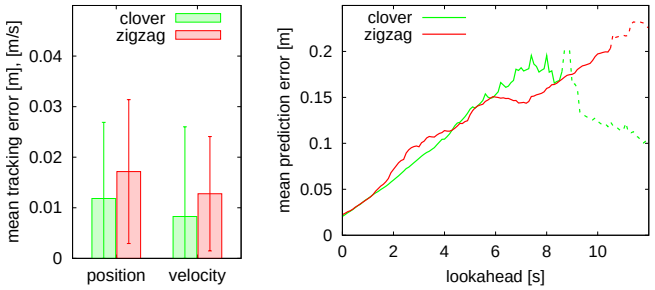

Fig. 7. *Left:* Average Euclidean distance between the planned and actual values for position and translational velocity of the robot in the obstacle courses shown in Fig. 6. The errorbars show the standard deviation. *Right:* Average Euclidean distance between the predicted and the actual position of the robot, as a function of the temporal lookahead added to the planning delay t_{plan}. In the dashed parts, less than 20 values contributed to the mean.

search in a finely discretized version of the parameter space. In all cases, the parameters of the trajectory generated by our optimization approach fall into the discretization bin of the parameters determined by the exhaustive search, and the trajectories yield the same time of travel.

In analysis, all pairs of parameters showed smooth and convex optimization manifolds, which supports the assumption that the parameters may be optimized independently.

Usually, the optimization achieves the major improvements in the first 50 iterations, and 300 iterations are sufficient to reach the optimum. We choose $t_{\text{plan}} = 0.4\,s$ for our experiments, which roughly corresponds to 400 iterations. On average, the optimization reduces the estimated traveling time by 31% compared to the initial trajectory.

B. Motion planning in obstacle courses

This experiment tests the precision and predictiveness of our motion planning system. We use a Pioneer P3-DX robot equipped with a subnotebook (Intel Core 2 Duo, 1.6 GHz) for all computations and a laser range finder for localization and obstacle detection. The robot drives with a velocity limit of 0.5 m/s. The error-feedback controller, the laser scanner and the odometry are all running at 35 Hz. We have set up two obstacle courses and marked a sequence of goals for the global path planner in the map. The path planner switches to the next goal whenever the robot has come closer than 1 m.

The trajectories of the robot are shown in Fig. 6. While the goal locations for the global path planner are fixed, replanning is executed periodically and the locations used for trajectory planning are not spatially aligned. Thus, the positions of pruned waypoints, which are input to the trajectory planner, vary between the rounds. Nevertheless, the trajectories from the different rounds are very similar, which indicates that a) the parameter set for the optimization, which contains tangent elongation and movement of waypoints, is appropriate to find a global optimum, and b) the optimization generates reproducible trajectories that are not critically dependent on the input waypoints.

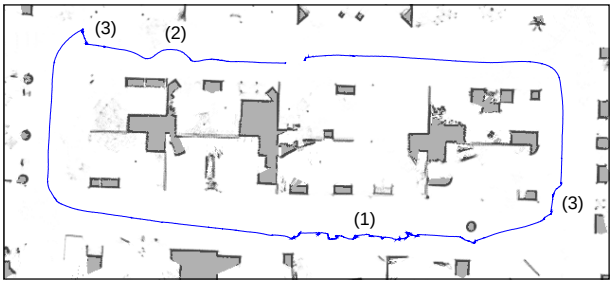

Fig. 8. Sample trajectory of our tour-guide robot "Albert" at a trade show. One person was blocking the robot's path on purpose for a longer period (1). However, the motion planning system plans smooth trajectories around unmapped obstacles if seen in advance (2), but can also execute sharp turns if required, i.e., if an obstacle suddenly occurs in front of the robot (3). The frame covers an area of 33m x 15m.

The average positional tracking error, i.e., how far the robot deviates from the planned path according to odometry, is around 1 cm for both obstacle courses, and the average deviation in velocity is below 2 cm/s, as shown in Fig. 7 (left). Both errors are a little higher in the zigzag course which has tighter curves.

The planned trajectories can be used to predict the robot's position over time in global coordinates. The corresponding error is shown in Fig. 7 (right) as a function of a lookahead time added to t_{plan}. Thus, a lookahead of 0 s corresponds to predictions over t_{plan}, which are needed for replanning.

Without replanning, the prediction error would reduce to the tracking error plus the accumulated error in the odometry readings. With replanning, as in the experiments, the predicting trajectory is seamlessly replaced after one second at most. Still, the mean prediction error is small, e.g., only around 15 cm for a lookahead of 6 s. This shows that replanning does not cause abrupt changes in the planned trajectory. Human observers can actually not recognize when switching of replanned trajectories takes place. Naturally, changes of the global goal location falsify these values and are filtered out, which causes a low number of values contributing to the averages for lookahead times greater than 8 s in the figure.

C. Navigation in populated environments

To evaluate the performance of our motion planning system in an application with dynamic obstacles, we tested the system on our tour-guide robot "Albert" during a three-day trade show in Freiburg. Motion planning was executed on the same subnotebook, with the odometry and the error-feedback controller running at 10 Hz. An example trajectory is shown in Fig. 8: the system is able to evade obstacles with smooth trajectories, but if necessary, sharp turns can be generated as well. The performance in presence of a person who deliberately blocks the robot's path could be improved by adding an additional strategic planner to avoid oscillation

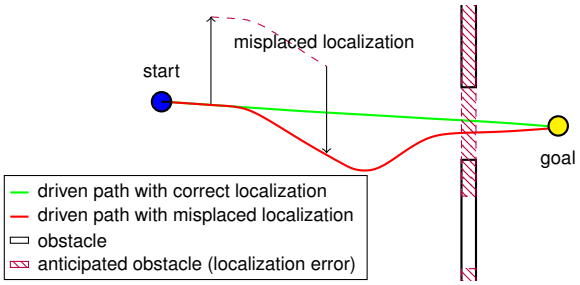

Fig. 9. Experimental result with misplaced self-localization. All course corrections are made by attaching new trajectory pieces to the existing one without violating the requirements on smoothness and curvature continuity.

of plans generated by the global path planner when the person moves from one side to the other.

D. Recovery from localization errors

The optimization process moves waypoints of trajectories, considering the location of both mapped and unmapped obstacles. In this way, our motion planning system can compensate for errors in localization to a certain degree. If however the localization was significantly misplaced, e.g., such that the anticipated and true position of free passages between obstacles do not overlap (as in Fig. 9), the system would naturally have problems to find its way. Since localization errors can be of temporary nature, we require our system to recover gracefully, i.e., with smooth trajectories, after the localization has been corrected. This is achieved by periodic replanning of the trajectories. The reaction of our system to such a localization failure is shown in Fig. 9.

In the run with correct localization, marked green in the figure, the robot passes the obstacles on a straight path. In the other run, marked red, the localization is artificially misplaced by 1.8 m shortly after the start. The robot adapts its trajectory in the next replanning step and heads for the supposed opening between the wrongly anticipated obstacles. After the localization recovers, the robot corrects its trajectory in the subsequent replanning step and drives to the goal.

VII. CONCLUSION

In this paper, we presented an approach to kinodynamic motion planning that determines time-optimal motion trajectories via optimization, starting from a given straight-line path. These trajectories are represented as joined quintic Bézier spline segments to achieve curvature continuity, in the middle of the segments as well as at the join points. Existing trajectories can be extended or replaced with newly generated pieces in order to account for unmapped obstacles. This replanning aspect also allows for graceful recovery in the case of localization discontinuities, and prevents accumulation of odometry errors.

The approach has been implemented and tested on real robots in complex and populated environments. Our experiments show that our system plans motion trajectories that can be executed precisely, i.e., with very low deviations from planned positions and velocities. Additionally, it can effectively replan if the robot encounters unexpected obstacles, for example in populated environments. Finally, as the motions are planned over time, the robot can predict its own motion with high precision.

REFERENCES

[1] J.-C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.
[2] S. Thrun and A. Bucken, "Learning maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, pp. 21–71, 1998.
[3] S. Koenig and M. Likhachev, "D* Lite," in *Proceedings of the AAAI Conference of Artificial Intelligence (AAAI)*, 2002.
[4] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm," in *Proc. of the Intl. Conf. on Automated Planning and Scheduling (ICAPS)*, 2005.
[5] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Intl. Journal of Robotics Research*, vol. 5, pp. 90–98, 1986.
[6] J. Borenstein and Y. Koren, "The vector field histogram - fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.
[7] J. Minguez and L. Montano, "Nearness diagram navigation (ND): Collision avoidance in troublesome scenarios," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 1, pp. 45–59, 2004.
[8] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.
[9] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the Association for Computing Machinery*, vol. 40, no. 5, pp. 1048–1066, November 1993.
[10] C. Stachniss and W. Burgard, "An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and System*, vol. 1, 30 Sept.–5 Oct. 2002, pp. 508–513.
[11] C. Mandel and U. Frese, "Comparison of wheelchair user interfaces for the paralysed: Head-joystick vs. verbal path selection from an offered route-set," in *3rd European Conference on Mobile Robotics*, 2007.
[12] A. Sahraei, M. T. Manzuri, M. R. Razvan, M. Tajfard, and S. Khoshbakht, *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*. Springer, 2007, ch. Real-Time Trajectory Generation for Mobile Robots, pp. 459–470.
[13] T. M. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *The International Journal of Robotics Research*, vol. 26, pp. 141–166, 2007.
[14] M. Likhachev and D. Ferguson, "Planning long dynamically-feasible maneuvers for autonomous vehicles," in *Robotics: Science and Systems Conference*, Zurich, 2008.
[15] S. Gulati and B. Kuipers, "High performance control for graceful motion of an intelligent wheelchair," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2008.
[16] Z. Shiller and Y. Gwo, "Dynamic motion planning of autonomous vehicles," *IEEE Transactions on Robotics and Automation*, vol. 7, pp. 241–249, 1991.
[17] K. Macek, D. Vasquez, T. Fraichard, and R. Siegwart, "Safe vehicle navigation in dynamic urban scenarios," in *Proc. 11th IEEE Intl. Conf. on Intelligent Transportation Systems (ITSC)*, 2008, pp. 482–489.
[18] 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, pp. 787–791.
[19] J. Connors and G. Elkaim, "Manipulating B-Spline based paths for obstacle avoidance in autonomous ground vehicles," in *Proc. of the ION National Technical Meeting*, San Diego, California, 2007.
[20] J. Loustau and M. Dillon, *Linear Geometry with Computer Graphics*. CRC Press, 1992.
[21] R. Plato, *Concise Numerical Mathematics*, ser. Graduate studies in mathematics. American Mathematical Society, 2003, vol. 57.
[22] C. Sprunk, "Planning motion trajectories for mobile robots using splines," University of Freiburg, 2008, http://www.informatik.uni-freiburg.de/~lau/students/Sprunk2008.pdf.
[23] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *IEEE Intl. Conf. on Neural Networks*, San Francisco, CA, 1993, pp. 586–591.
[24] G. Oriolo, A. D. Luca, and M. Vendittelli, "WMR control via dynamic feedback linearization: Design, implementation, and experimental validation," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 6, pp. 835–852, November 2002.