Constrained Global Path Optimization for Articulated Steering Vehicles

Ji-wung Choi, *Member, IEEE*, and Kalevi Huhtala

***Abstract*—This paper proposes a new efficient path-planning algorithm for articulated steering vehicles operating in semi-structured environments, in which obstacles are detected online by the vehicle's sensors. The first step of the algorithm is offline and computes a finite set of feasible motions that connect discrete robot states to construct a search space. The motion primitives are parameterized using Bézier curves and optimized as a nonlinear programming problem (NLP) equivalent to the constrained path planning problem. Applying the A^* search algorithm to the search space produces the shortest paths as a sequence of these primitives. The sequence is drivable and suboptimal, but it can cause unnatural swerves. Therefore, online path smoothing, which uses a gradient-based method, is applied to solve another NLP. Numerical simulations demonstrate that performance of the proposed algorithm is significantly better than that of existing methods when determining constrained path optimization. Moreover, field experimental results demonstrate the successful generation of fast and safe trajectories for real-time autonomous driving.**

***Index Terms*—Nonholonomic constraints, obstacle avoidance, optimization, path planning.**

I. Introduction

THE versatility of autonomous vehicles in academic, industrial, and military applications will have a profound effect in the future. Autonomous ground vehicles (AGVs), when integrated into the general public's vehicle-centric lifestyle, have some promising applications, such as autonomous parking [1]–[3], transportation [4], [5], and planetary exploration [6].

A. Motivation

This paper was motivated by the potential for autonomous wheeled loaders to carry pallets in ports, construction sites, factories, and mining fields. For this to be a viable option, a safe motion path must be generated in real time. This means that, to pick up a pallet, a robot should be able to reach directly forward and align itself with the pallet's fork holes while avoiding obstacles.

Therefore, the autonomous motion planning system should address not only global path planning but local obstacle avoidance as well. The development of wireless technology and the ability to gather and compute environmental data enable AGVs to easily obtain the information required for global path planning, such as global positioning system (GPS) coordinates, building blueprints, and vehicle guide paths. Precomputed information produces a predetermined global environment. However, a vehicle's local environment is unknown and dynamically changing due to the vehicle's limited sensor range. This means that obstacle avoidance must be quickly calculated via an online replanning process. One of the greatest challenges in autonomous motion planning is determining a balance between the computational demands of perceptive intelligence at the local level and the deliberative intelligence at the global level.

A common approach to this problem employs a hierarchical motion-planning architecture [6], [7]. Globally, four- or eight-connected grids create piecewise linear paths but ignore the vehicle's mobility constraints. Locally, on the other hand, a higher dimensional search space produces feasible paths while addressing these constraints. However, this planning architecture may not apply to cluttered environments because the motion fidelity differential between the global and local planners produces cross-track errors that can lead to collisions. Moreover, even global planners must occasionally produce high-fidelity motions, such as switching direction in cluttered areas. This paper focuses on improving both global motion fidelities and computational efficiency for the operation of AGVs in cluttered environments.

B. Related Works

Most current algorithms produce computationally efficient motion for discrete state spaces [6], [8], [9]. However, their resulting global paths are generally not smooth (piecewise linear), which means the kinematics are unattainable for nonholonomic mobile robots. Although adapting a higher resolution local planner that accounts for a robot's mobility constraints can overcome this drawback, the possibility of collisions between the robot and obstacles remains when operating in cluttered environments.

As a result, path-smoothing methods are necessary, such as the well-known Dubins path method. Dubins demonstrated that the shortest paths for car-like robots consist of two circular arcs and line segments [10]. Although this method has been widely used and extended by many researchers [11], [12], the generated paths contain discontinuous curvature at the joint nodes that connect the lines and arcs. These paths require vehicles to stop and reposition to alter their steering angles. Therefore, curvature continuity is an important requirement for trajectory guidance applications. To remedy this problem, Kanayama and Miyake [13] employed clothoids to smooth the joint nodes that connect the lines to the arcs. The resulting path is C^2

Manuscript received May 1, 2014; revised January 3, 2015; accepted March 29, 2015. Date of publication April 30, 2015; date of current version April 14, 2016. The review of this paper was coordinated by Prof. J. Wang.

J. Choi is with the Intelligent Vehicle Research Center, Hyundai MOBIS Company, Ltd., Uiwang 437-815, Korea (e-mail: jiwung.choi@mobis.co.kr).

K. Huhtala is with the Department of Intelligent Hydraulics and Automation, Tampere University of Technology, 33101 Tampere, Finland (e-mail: kalevi.huhtala@tut.fi).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVT.2015.2424933

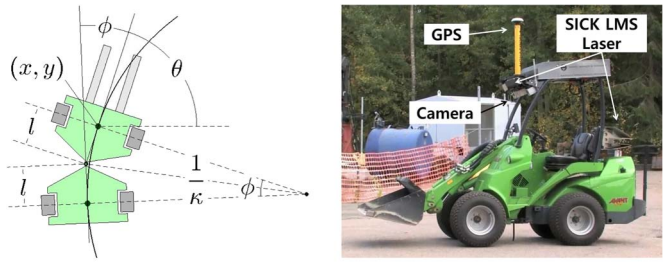
continuous but produces another problem: Clothoids cannot be expressed in closed form, which leads to computational difficulties. Lamiraux and Laumond [14] revisited the problem, considering the robot as a 4-D system from a kinematic point of view and as a 3-D system from a geometric point of view to compute a collision-free C^2 path. Fraichard and Scheuer [15] extended the method used in [14] and accounted for the curvature's upper bound and the robot's curvature derivative constraints. Maekawa *et al.* [5] used a B-spline to produce collision-free curvature-continuous paths, which contained shorter lengths than the method used in [15]. However, their proposed approach did not effectively address obstacle avoidance because control points were manually moved until the path missed the obstacles.

The proposed approach of this paper adapts quintic Bézier curves to parameterize a path in a compact form. Quintic Bézier curves have been widely utilized to produce curvature-continuous paths because they smooth straight-line paths [16], [17]. Lau *et al.* used curves that connect two consecutive waypoints to generate time-optimal trajectories under a mobile robot's kinodynamic constraints [16]. Choi *et al.* added a curve to the corner area around a waypoint [17]. Their numerical simulations demonstrate the superiority of their method's resulting smoothness to the smoothness produced using the method presented in [16]. Whereas the focus in [17] is limited to structured environments, such as highways and racecourses, this paper extends their research to unstructured cluttered environments.

Optimal control problems can produce robust solutions. Representation of the performance index depends on a user's interests, such as energy consumption [18], travel time [19], [20], smoothness (evaluated by the integral of the curvature-squared function) [3], [5], [17], [21], or safety from collisions (evaluated by the integral of the proximity-to-obstacle function) [3], [17]. Kelly and Nagy [21] optimize parametric smoothness motions using cubic polynomial curvature functions of the path lengths. Their work has been successfully implemented in motion planning for driverless cars [2], [22]. However, a drawback to this explicit control function is the need for solution state integrals to be evaluated by decoupling nonlinear differential equations. The computational burden increases with an increase in the number of primitives that comprise the global paths. Overall, global path smoothing has not been considered in this method. Instead, local trajectories have been repeatedly generated to converge to a global trajectory. Although these generated paths are drivable and suboptimal, they can contain unnatural swerves that require unnecessary steering [3], particularly in a complex environment. Dolgov *et al.* [3] optimized the coordinates of a global path's vertices. However, their method requires an extra step to interpolate the optimized coordinates because following the sequence of reduced-dimensional states is not possible.

C. Problem Statement

The path-planning problem addressed in this paper is formulated as follows. Because the robot must reach directly forward and be aligned with a pallet's fork holes, the dimension of the vehicle state vector $\mathbf{q}$ must be at least four, accounting for reference position (x, y), orientation θ, and curvature κ. Control vector $\mathbf{u}$ consists of two dimensions: the longitudinal velocity v



(a) (b)

Fig. 1. (a) Kinematic model of articulated steering vehicles (ASVs). (b) GIM, which is an autonomous ASV from Tampere University of Technology (TUT) equipped with two LIDAR units, a high-accuracy GPS, a camera sensor, an inertial-measurement system, and a switchable front-end effector between the buckets and folks.

and the angular acceleration σ, which is related to the steering velocity $\dot{\phi}$. Then, the motion is determined as follows:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\kappa} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ \kappa \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \sigma. \tag{1}$$

The proposed method's platform is an articulated steering-type vehicle (ASV). The ASV steers by actuating the center joint between the front and rear half bodies, shown in Fig. 1(a), which leads to the following relationship:

$$\kappa = \frac{\tan\frac{\phi}{2}}{l} \quad \sigma = \dot{\kappa} = \frac{\dot{\phi}}{2l\cos^2\frac{\phi}{2}} \tag{2}$$

in which l is the distance between an axle and center joint. This path-planning algorithm is not restricted to ASVs. However, (2) has a different representation depending on the wheel steering mobility system being used. Fig. 1(b) shows the robot used in this paper, which is a generic intelligent machine (GIM), which is equipped with two light detection and ranging (LIDAR) units for forward and backward fields of view, a high-accuracy GPS, and a camera sensor.

The focus of this paper is on path planning. Therefore, a constant driving velocity is assumed, i.e.,

$$|v| = v_{\text{constant}}.$$

The resulting path must be constrained by a bounded curvature, i.e.,

$$|\kappa| \leq \kappa_{\max} = \frac{1}{l}\tan\frac{\phi_{\max}}{2} \tag{3}$$

where $\phi_{\max}$ is the mechanically limited steering angle.

Let $\mathcal{C}_{\text{free}}$ be the free space in a configuration space. Given the vehicle's initial state $\mathbf{q}_{\text{init}}$ and goal state $\mathbf{q}_{\text{goal}}$, the path planner produces a free path Γ, i.e.,

$$\Gamma : [0,1] \to \mathcal{C}_{\text{free}}$$

subject to the robot's kinematic constraints (1), (3), and the boundary constraints

$$\Gamma(0) = \mathbf{q}_{\text{init}}, \quad \Gamma(1) = \mathbf{q}_{\text{goal}}. \tag{4}$$

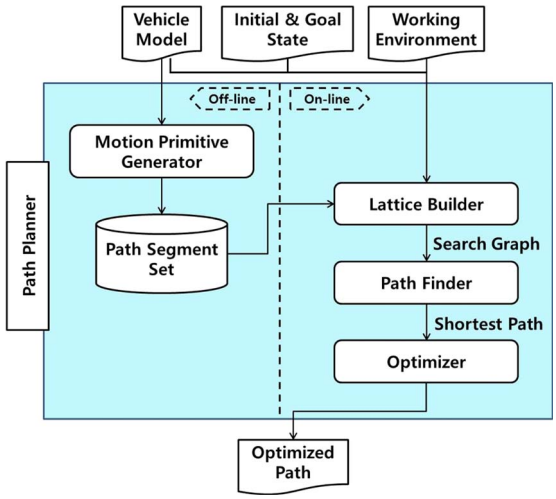


Fig. 2. Path-planning system architecture.

D. System Architecture

The proposed solution to the path-planning problem uses the system architecture presented in Fig. 2. The vehicle model, the initial and goal state boundary pair, and working environment define the inputs for the path planner, whereas the output is an optimized path. The path planner is divided into two subsystems, depending on whether the computation is performed online or offline. The first subsystem is the motion primitive generator, which computes a finite set of feasible path segments subject to the vehicle model's constraints offline (see Section II). The second subsystem operates online to handle the dynamically changing inputs: the initial state, goal state, and sensed environment. It can be conceptualized as a three-level hierarchy: 1) The *lattice builder* constructs a search graph by connecting the precomputed path segments (edges) to the discrete robot states (vertices) that were sampled from the sensed environment in Section III; 2) the *path finder* applies the A^* algorithm to the search graph to determine the shortest path; and 3) the *optimizer* smoothes the sequence of path segments by solving the constrained optimization problem in Section IV.

E. Contributions

This paper demonstrates significant path-planning contributions for ASVs operating in a semi-structured environment and can be summarized follows.

- The proposed path-planning algorithm is computationally efficient and produces high-fidelity global paths in real time. While most current work [6], [8], [9] does not address a vehicle's mobility constraints in the global planning process for computational reasons, the proposed algorithm not only accounts for mobility constraints but effectively refines the feasible path by solving the constrained optimization problem in Section IV. The global path optimization process involves arc lengths, quadratic curvatures, and proximity to obstacles to generate smoother, safer robot motions, as is demonstrated by the experimental results. Of greatest significance, planning computations were sufficiently fast, enabling online replanning in a semi-structured environment. Numerical simulations demonstrate a remarkably improved performance when compared with the existing work of Dolgov *et al.* [3]. This is shown in Section V.
- Quintic Bézier curves are used to construct paths, and parametric Bézier curves are beneficial when solving constrained optimization problems for the following reasons (see Section IV).
  - — Feasible motions are spanned using few parameters.
  - — States along Bézier curves are expressed in closed form, whereas widely used polynomial curvature functions [2], [21], [22] must evaluate solution state integrals by decoupling nonlinear differential equations.
  - — The initial and goal state boundary constraints are easily satisfied using control points so that, given the typical form of performance index $J = \Phi(\mathbf{q}(t_f)) + \int_{t_0}^{t_f} L(\mathbf{q}, \mathbf{u}, t)dt$, the terminal error penalty function $\Phi(\mathbf{q}(t_f))$ is eliminated.
  - — The gradient of the performance index involving arc lengths, smoothness, and collision clearance is efficiently computed.
  - — The differential constraint can be satisfied without the use of an interpolation process.
  - — A path's direction-switching points can be deformed and processed in a one-stage optimization process.
- A new type of distance map, which is called the *bidirectional transformed distance map*, is proposed to effectively avoid collisions. Typical distance maps are constructed by computing distance transforms from obstacle grid cells through free-space cells and do not contain distance fields in the obstacle cells. Therefore, they may not apply to reactive path optimization, particularly when the initially planned paths involve collisions with obstacles that are detected while the robot follows the path. On the other hand, a bidirectional distance map is constructed by computing distance transforms in both directions, between the obstacle and free cells, and effectively deforms paths to avoid obstacles. (See Section IV-A for more details.)

II. Generation of Motion Primitives

The first step in our path-planning algorithm generates the feasible motion primitives that construct a global path. This section presents a method for computing motion primitives using quintic Bézier curves.

A. Bézier Curve Parametric Path

A Bézier curve of degree n, i.e., $\mathbf{B}(t)$, is defined by $n+1$ control points $\mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_n$ as shown in the following:

$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{B}_i, \quad t \in [0,1]. \tag{5}$$

The following are some useful properties of Bézier curves that can be applied to path planning.

- Bézier curves always begin at $\mathbf{B}_0$ and end at $\mathbf{B}_n$, i.e.,

$$\mathbf{B}(0) = \mathbf{B}_0, \quad \mathbf{B}(1) = \mathbf{B}_n. \tag{6}$$

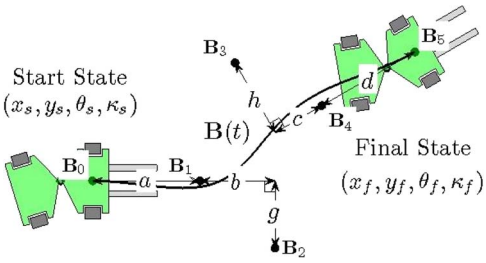


Fig. 3. Quintic Bézier curve $\mathbf{B}(t)$ can be determined using 12-tuple $\lambda = \{x_s, y_s, \theta_s, \kappa_s, a, b, c, d, x_f, y_f, \theta_f, \kappa_f\}$, where $(x_s, y_s, \theta_s, \kappa_s)$ is the initial state, $(x_f, y_f, \theta_f, \kappa_f)$ is the final state, and (a, b, c, d) is the control distance vector.

- The orientations at their endpoints are

$$\theta(0) = \operatorname{atan2}\left(\mathbf{B}_{1_y} - \mathbf{B}_{0_y}, \mathbf{B}_{1_x} - \mathbf{B}_{0_x}\right)$$
$$\theta(1) = \operatorname{atan2}\left(\mathbf{B}_{n_y} - \mathbf{B}_{n-1_y}, \mathbf{B}_{n_x} - \mathbf{B}_{n-1_x}\right). \quad (7)$$

- The curvatures at their endpoints are given by

$$\kappa(0) = \frac{(n-1)g}{n\|\mathbf{B}_1 - \mathbf{B}_0\|^2}, \quad \kappa(1) = \frac{(n-1)h}{n\|\mathbf{B}_n - \mathbf{B}_{n-1}\|^2} \quad (8)$$

  where g is the perpendicular distance from $\mathbf{B}_2$ to $\mathbf{B}_1 - \mathbf{B}_0$, and h is similarly defined. An illustration of this is shown in Fig. 3.

It should be noted that the compactness of curve parameters strongly influences optimality and convergence rate when optimizing piecewise curves. Details of this are found in Section IV. Hence, smoothly joining low-degree Bézier curves are preferable for path planning. The proposed method uses quintic Bézier curves because path planning fundamentally requires a path through the beginning and end points using specified orientations and curvatures, and a degree of five is necessary to satisfy this requirement. Justification for this assertion follows.

To generate a motion segment, in the form of a quintic Bézier curve, which connects the two states $\mathbf{q}_s = (x_s, y_s, \theta_s, \kappa_s)^T$ and $\mathbf{q}_f = (x_f, y_f, \theta_f, \kappa_f)^T$, which is shown in Fig. 3, given the end-point property in (6), the first and the last control points must be located at the two end positions, i.e.,

$$\mathbf{B}_0 = \mathbf{p}_s = (x_s, y_s)^T, \quad \mathbf{B}_5 = \mathbf{p}_f = (x_f, y_f)^T. \quad (9)$$

The tangential property in (7) implies that the first two and last two control points must align with the end orientations, i.e.,

$$\mathbf{B}_1 = \mathbf{p}_s + a\hat{\theta}_s, \quad \mathbf{B}_4 = \mathbf{p}_f - d\hat{\theta}_f, \quad a, d \in \mathbb{R} \quad (10)$$

in which $\hat{\theta}_s$ and $\hat{\theta}_f$ represent the unit vectors corresponding to θ_s and θ_f, respectively, i.e.,

$$\hat{\theta}_s = (\cos\theta_s, \sin\theta_s)^T, \quad \hat{\theta}_f = (\cos\theta_f, \sin\theta_f)^T.$$

Here, a and d are the distances between the control points in their respective orientations, which are shown in Fig. 3 and defined as follows:

$$a = (\mathbf{B}_1 - \mathbf{B}_0) \cdot \hat{\theta}_s, \quad d = (\mathbf{B}_4 - \mathbf{B}_5) \cdot \hat{\theta}_f.$$

In addition, according to the end curvature property in (8), $\mathbf{B}_2$ is constrained to be on the line parallel to $\mathbf{B}_1 - \mathbf{B}_0$ with distance $g = (5/4)a^2\kappa_s$, and $\mathbf{B}_3$ is similarly retrospectively constrained, as shown in Fig. 3. That is

$$\mathbf{B}_2 = \mathbf{p}_s + (a+b)\hat{\theta}_s + \frac{5}{4}a^2\kappa_s\dot{\hat{\theta}}_s, \quad b \in \mathbb{R}$$
$$\mathbf{B}_3 = \mathbf{p}_f - (c+d)\hat{\theta}_f + \frac{5}{4}d^2\kappa_f\dot{\hat{\theta}}_f, \quad c \in \mathbb{R}. \quad (11)$$

Because a quintic Bézier curve is solely determined by six control points, representing the control points in a manner similar to (9)–(11) transforms the curve, allowing it to be spanned by the following 12-tuple parameter.

$$\lambda = (x_s \quad y_s \quad \theta_s \quad \kappa_s \quad a \quad b \quad c \quad d \quad x_f \quad y_f \quad \theta_f \quad \kappa_f)^T$$

Incorporating (9)–(11) into (5) with $n = 5$ creates a new representation of the curve:

$$\mathbf{B}(\lambda, t) = \mathbf{F}(\lambda)\mathbf{G}(t) \quad (12)$$

in which

$$\mathbf{F}(\lambda) = (\mathbf{p}_s \; a^2\kappa_s\dot{\hat{\theta}}_s \; a\hat{\theta}_s \; b\hat{\theta}_s \; c\hat{\theta}_f \; d\hat{\theta}_f \; d^2\kappa_f\dot{\hat{\theta}}_f \; \mathbf{p}_f) \quad (13)$$

$$\mathbf{G}(t) = \begin{pmatrix} (1-t)^3(6t^2+3t+1) \\ \frac{25}{2}t^2(1-t)^3 \\ 5t(1-t)^3(t+1) \\ 10t^2(1-t)^3 \\ -10t^3(1-t)^2 \\ 5t^3(1-t)(t-2) \\ \frac{25}{2}t^3(1-t)^2 \\ t^3(6t^2-15t+10) \end{pmatrix}. \quad (14)$$

It should be noted that (12) divides Bézier curve $\mathbf{B}$ into two matrices: $\mathbf{F}$, which depends on path parameter λ, and $\mathbf{G}$, which depends on intrinsic Bézier parameter t. Using a form similar to (12) is beneficial when solving NLPs involving $\mathbf{B}$ for multiple reasons.

- It is efficient to compute the gradient of the performance index that involves some states on the curve, such as arc length or curvature. Because the states are determined by $\mathbf{B}(t)$ or $\dot{\mathbf{B}}(t)$, the gradient is determined using the respective partial derivative of $\mathbf{B}(t)$ or $\dot{\mathbf{B}}(t)$ with respect to $\mathbf{X}$, the solution variable of NLP, which is independent of t but dependent on λ. Therefore, the representation used in (12) enables an easy computation of the partial derivatives in (16) and (17).
- Compactly parameterizing piecewise Bézier curve paths is efficient. Naive control point parametric curves require some geometric equality constraints for the G^2 continuity constraint imposed on two adjacent path segments [17]. However, the proposed parametric curve in (12) effectively satisfies the G^2 constraint by merging the configurations that join adjacent curves, thereby reducing the path parameter's size without requiring any equality constraints. A more detailed discussion of this can be found in Section IV.

As a result, the proposed parametric curve (12) produces greater optimality and increases computation speed when solving NLPs.

Differentiating (12) with respect to t yields the following derivatives of $\mathbf{B}(t)$:

$$\dot{\mathbf{B}}(t) = \mathbf{F}(\lambda)\dot{\mathbf{G}}(t) \tag{15}$$

Defining $\mathbf{F}_x(\lambda)$ and $\mathbf{F}_y(\lambda)$ to be the respective first and second row vectors of $\mathbf{F}(\lambda)$ results in the following partial derivatives of $\mathbf{B}$ and $\dot{\mathbf{B}}$ as follows:

$$\frac{\partial \mathbf{B}_x(t)}{\partial \lambda} = \dot{\boldsymbol{F}}_x(\lambda)\mathbf{G}(t), \quad \frac{\partial \mathbf{B}_y(t)}{\partial \lambda} = \dot{\boldsymbol{F}}_y(\lambda)\mathbf{G}(t) \tag{16}$$

$$\frac{\partial \dot{\mathbf{B}}_x(t)}{\partial \lambda} = \dot{\boldsymbol{F}}_x(\lambda)\dot{\mathbf{G}}(t), \quad \frac{\partial \dot{\mathbf{B}}_y(t)}{\partial \lambda} = \dot{\boldsymbol{F}}_y(\lambda)\dot{\mathbf{G}}(t). \tag{17}$$

Here, $\dot{\boldsymbol{F}}_x(\lambda)$ and $\dot{\boldsymbol{F}}_y(\lambda)$ are obtained by differentiating (13) with respect to λ and are represented as the equation shown at the bottom of the page.

B. State Sampling

In conventional optimization problems, a performance index is evaluated by integrating some function of the solution parameters. Numerical implementation requires the discretization of states on the Bézier curves using discrete parameter $\{t_0, \ldots, t_k\} \subset [0, 1]$. Incorporating (12) and (15) with parameter t_j allows for an extraction of the sampled positions and orientations, i.e.,

$$x_j = \mathbf{B}_x(t_j), \quad y_j = \mathbf{B}_y(t_j), \quad \theta_j = \mathrm{a\tan 2}\left(\dot{\mathbf{B}}_y(t_j), \dot{\mathbf{B}}_x(t_j)\right). \tag{18}$$

Moreover, to evaluate a trajectory's smoothness, the discrete curvature is utilized and can be approximated using the following:

$$\kappa_j = \mathrm{sgn}\left(\dot{\mathbf{B}}(t_j) \times \dot{\mathbf{B}}(t_{j+1})\right) \frac{\Delta\theta_j}{s_j} \tag{19}$$

where $\Delta\theta_j$ is the difference between θ_j and θ_{j+1} given by

$$\Delta\theta_j = \cos^{-1}\left(\frac{\dot{\mathbf{B}}(t_j) \cdot \dot{\mathbf{B}}(t_{j+1})}{\left\|\dot{\mathbf{B}}(t_j)\right\| \left\|\dot{\mathbf{B}}(t_{j+1})\right\|}\right)$$

and s_j is the distance between $(x_j, y_j)^T$ and $(x_{j+1}, y_{j+1})^T$ and is defined as follows:

$$s_j = \left\|\mathbf{B}(t_{j+1}) - \mathbf{B}(t_j)\right\|. \tag{20}$$

C. Optimization for Boundary Value Problems

Parametric Bézier curves are solutions to the boundary value problem and are used to generate feasible motions that drive a robot from one state value to another. As described earlier, the parametric Bézier curve $\mathbf{B}(\lambda, t)$, initiating at $\mathbf{q}_s$ and finalizing at $\mathbf{q}_f$, has four degrees of freedom, which are spanned by $\mathbf{h} = (a, b, c, d)^T$. Given two end states, the optimal control distance vector $\mathbf{h}^*$, which generates short and smooth trajectories under the robot's maximum curvature constraint, must be determined and can be formulated as the following NLP problem.

$$\dot{\boldsymbol{F}}_x(\lambda) = \begin{pmatrix} \partial\mathbf{F}_x/\partial x_s \\ \partial\mathbf{F}_x/\partial y_s \\ \partial\mathbf{F}_x/\partial\theta_s \\ \partial\mathbf{F}_x/\partial\kappa_s \\ \partial\mathbf{F}_x/\partial a \\ \partial\mathbf{F}_x/\partial b \\ \partial\mathbf{F}_x/\partial c \\ \partial\mathbf{F}_x/\partial d \\ \partial\mathbf{F}_x/\partial x_f \\ \partial\mathbf{F}_x/\partial y_f \\ \partial\mathbf{F}_x/\partial\theta_f \\ \partial\mathbf{F}_x/\partial\kappa_f \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -a^2\kappa_s\cos\theta_s & -a\sin\theta_s & -b\sin\theta_s & 0 & 0 & 0 & 0 \\ 0 & -a^2\sin\theta_s & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2a\kappa_s\sin\theta_s & \cos\theta_s & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\theta_s & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cos\theta_f & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cos\theta_f & -2d\kappa_f\sin\theta_f & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -c\sin\theta_f & -d\sin\theta_f & -d^2\kappa_f\cos\theta_f & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -d^2\sin\theta_f & 0 \end{pmatrix}$$

$$\dot{\boldsymbol{F}}_y(\lambda) = \begin{pmatrix} \partial\mathbf{F}_y/\partial x_s \\ \partial\mathbf{F}_y/\partial y_s \\ \partial\mathbf{F}_y/\partial\theta_s \\ \partial\mathbf{F}_y/\partial\kappa_s \\ \partial\mathbf{F}_y/\partial a \\ \partial\mathbf{F}_y/\partial b \\ \partial\mathbf{F}_y/\partial c \\ \partial\mathbf{F}_y/\partial d \\ \partial\mathbf{F}_y/\partial x_f \\ \partial\mathbf{F}_y/\partial y_f \\ \partial\mathbf{F}_y/\partial\theta_f \\ \partial\mathbf{F}_y/\partial\kappa_f \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -a^2\kappa_s\sin\theta_s & a\cos\theta_s & b\cos\theta_s & 0 & 0 & 0 & 0 \\ 0 & a^2\cos\theta_s & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2a\kappa_s\cos\theta_s & \sin\theta_s & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sin\theta_s & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sin\theta_f & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sin\theta_f & 2d\kappa_f\cos\theta_f & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & c\cos\theta_f & d\cos\theta_f & -d^2\kappa_f\sin\theta_f & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & d^2\cos\theta_f & 0 \end{pmatrix}.$$

Minimize

$$J(\mathbf{h}) = \sum_{j=0}^{k-1} \left[w_s \cdot s_j(\mathbf{h}) + w_\kappa \cdot \kappa_j^2(\mathbf{h}) \right] \tag{21}$$

subject to

$$\kappa_j^2 < \kappa_{\max}^2. \tag{22}$$

Performance index J consists of two terms, which produce short smooth trajectories by penalizing arc length and curvature, and are tuned using w_s and w_κ. The solution, with its maximum magnitude of curvature less than the robot's limit $\kappa_{\max}$ in (22) can be selected as a feasible motion segment.

The globally convergent method of moving asymptotes (MMA) algorithm [23] is adapted to solve the constrained optimization problem found in (21) and (22). It is widely known that the performance of gradient-based methods, including the MMA algorithm, heavily depends on two factors: the proximity of the initial guess to the optima and the computational efficiency of the gradient of the performance index. Empirically, one quarter of the length between $\mathbf{p}_s$ and $\mathbf{p}_f$ was determined to be a good initial prediction for each control distance, i.e.,

$$\mathbf{h}_{\text{guess}} = \|\mathbf{p}_f - \mathbf{p}_s\| \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{pmatrix}^T. \tag{23}$$

The gradient of performance index (21) is derived using the partial derivative of s_j and κ_j, i.e.,

$$\frac{dJ(\mathbf{h})}{d\mathbf{h}} = \sum_{j=0}^{k-1} \left[w_s \cdot \frac{\partial s_j}{\partial \mathbf{h}} + w_\kappa \cdot 2\kappa_j \frac{\partial \kappa_j}{\partial \mathbf{h}} \right]. \tag{24}$$

These derivatives are obtained by differentiating (18)–(20) with respect to λ as follows:

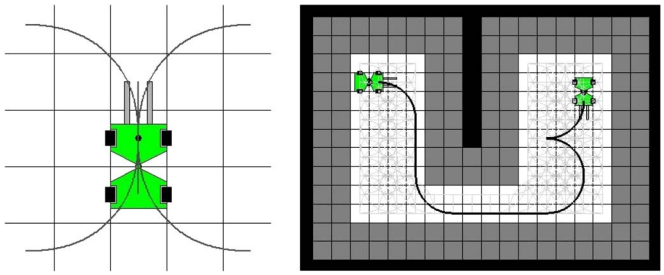
$$\frac{\partial \theta_j}{\partial \lambda} = \frac{\dot{\mathbf{B}}(t_j)}{\dot{\mathbf{B}}(t_j) \cdot \dot{\mathbf{B}}(t_j)} \times \frac{\partial \dot{\mathbf{B}}(t_j)}{\partial \lambda} \tag{25}$$

$$\frac{\partial \kappa_j}{\partial \lambda} = \frac{1}{s_j} \left[\frac{\partial \theta_{j+1}}{\partial \lambda} - \frac{\partial \theta_j}{\partial \lambda} - \kappa_j \frac{\partial s_j}{\partial \lambda} \right] \tag{26}$$

$$\frac{\partial s_j}{\partial \lambda} = \frac{\mathbf{B}(t_{j+1}) - \mathbf{B}(t_j)}{\|\mathbf{B}(t_{j+1}) - \mathbf{B}(t_j)\|} \cdot \left(\frac{\partial \mathbf{B}(t_{j+1})}{\partial \lambda} - \frac{\partial \mathbf{B}(t_j)}{\partial \lambda} \right). \tag{27}$$

III. Shortest Path Search in the State Lattice

The next step in the proposed method constructs a search graph using a finite set of the motion primitives generated earlier. Despite many external differences, the various path-planning algorithms can be formulated into graph searches, in which a finite set of robot states (vertices) are connected by feasible motion segments (edges). To construct the search graph, the *state lattice* paradigm is applied [24]. The state lattice method was proposed by Pivtoraiko and Kelly [24] to satisfy motion constraints while achieving the computational advantages of discretization. It is a graphical representation of a discretized configuration space, in which repeating pairs of vertices (states) are connected by a finite set of feasible motion segments, which are referred to as the *control set*. The method in Section II is presented as a means of generating the control set. Because each element in the set is subject to a robot's system constraints, such as its nonholonomic nature and maximum curvature, the constrained motion planning is formulated as a query: a graph search.



(a) (b)

Fig. 4. (a) Lattice composed of discretized states with a finite set of feasible motion segments designed to connect the discrete states. (b) Search graph constructed by copying the set to free space states.

Fig. 4 shows an example of the 3-D state lattice (x, y, θ). The position of (x, y) represents the center point for each grid, and orientation θ is spanned by $\{0, \pi/2, \pi, 3\pi/2\}$. Six solid curves, which are shown in Fig. 4(a), indicate the control set. The cells occupied by each curve are precomputed and translated into their appropriate states to expedite collision checking during the online planning. Although the generation of curvature continuous paths may seem to require 4-D state lattice (x, y, θ, κ), higher dimensional lattices produce increased computational complexity. Therefore, 3-D lattice (x, y, θ) with κ fixed at zero was used in this paper.

A search graph is constructed by copying the control set of states in a free space $\mathcal{C}_{\text{free}}$. The space is a complementary set of $\mathcal{C}_{\text{obstacle}}$, which is generated by inflating all obstacle cells using the robot's outer radius. In Fig. 4(b), the black and gray cells indicate obstacles and inflated space, respectively. Planning a motion that is relative to the obstacles equates to planning a motion of the robot's reference point relative to $\mathcal{C}_{\text{obstacle}}$. Therefore, a sequence of configurations in $\mathcal{C}_{\text{free}}$ of the graph forms a path that guarantees feasibility and collision avoidance. Applying A^* search algorithms to the graph produces the shortest path. The black curve in Fig. 4(b) is the solution found by applying A^*.

IV. Global Path Optimization

The global paths searched through the state lattice are not only feasible but suboptimal driving motions as well. However, they can produce unnatural swerves in a complex environment [3]. Therefore, further improvement in the solution's quality using nonlinear optimization is necessary and could produce a local (and frequently global) optimum.

A. Collision Avoidance

Global path optimization differs from motion primitive generation (see Section II) in that it takes obstacles into consideration. For the effective implementation of collision avoidance,

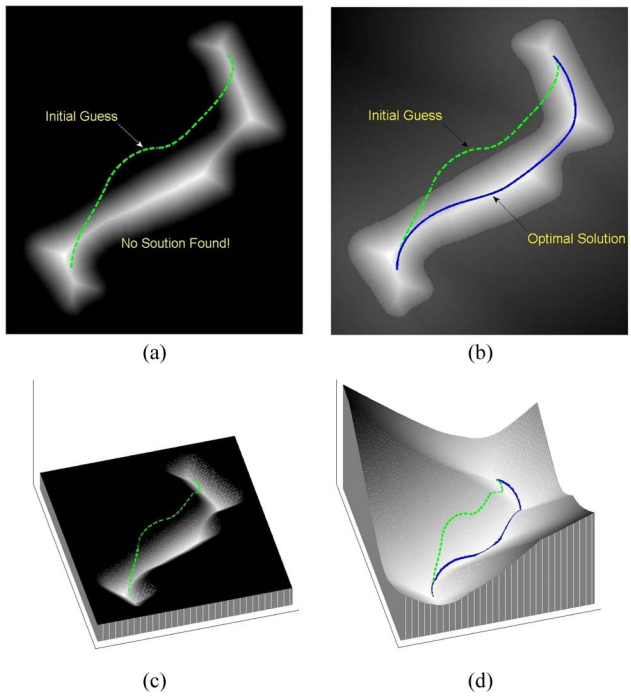


Fig. 5. (a) Typical distance map constructed by transforming distances from obstacles to free space. Maps label discrete grids with the distance to the nearest obstacle, indicated by darkness. (b) Bidirectional maps transform distance twice between the two spaces. (c) In the negative distance potential field of (a), the initial predicted path (dashed green curve) may create collisions with obstacles and may not converge to an optimum. (d) In the potential field of (b), the paths converge to the optimum (solid blue curve). (a) Typical distance transform. (b) Bidirectional transform. (c) Potential field of (a). (d) Potential field of (b).

a *distance map*, i.e., a map in which each discrete grid is labeled with the distance to the nearest obstacle, is used. Typical distance maps are constructed by computing the Euclidean distance transform from obstacle cells through free-space grid cells in a given environment [3], [25], [26]. However, the map may not suffice for reactive path optimization, particularly when the initially planned path creates collisions with obstacles that are detected while the path is being followed. New searches through the updated space provide a safe path that can be optimized but occasionally cause computational delays that require the robot to stop. To remedy this and reactively deform the colliding paths into an optimum, a *bidirectional transformed distance map* is proposed.

Fig. 5 compares the two distance maps. The typical distance map, which is shown in Fig. 5(a), transforms the distance from obstacle cells into free cells and labels the distance to the nearest obstacle, which is indicated by darkness. It should be noted that there is no distance field in the obstacle space (black fill). On the other hand, the bidirectional distance map in Fig. 5(b) transforms the distance twice: from obstacle cells to free cells with positive values and from free cells to obstacle cells with negative values. Consequently, obstacle cells also have distance fields that measure the closest free cells. Fig. 5(c) and (d) shows the negatively valued distance potential fields resulting from the two types of maps. The potential's slope in the obstacle space in Fig. 5(d) tends to depress the initially predicted path (dashed green curve) into the optimal solution (solid blue curve). However, Fig. 5(c) fails to find any solution using the same initial prediction.

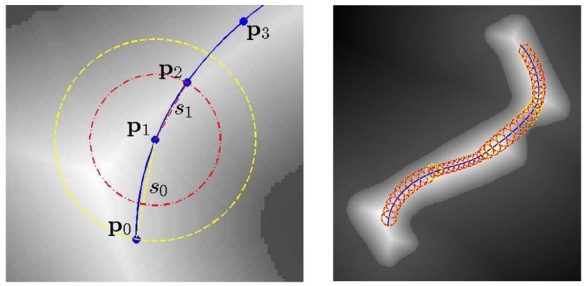


Fig. 6. Collision detection in a bidirectional transformed distance map. (a) If $\gamma(\mathbf{p}_1)$ is greater than $\|\mathbf{p}_1 - \mathbf{p}_0\|$, then every point on line segment $\mathbf{p}_1 - \mathbf{p}_0$ avoids obstacles. It also holds true between $\mathbf{p}_1$ and $\mathbf{p}_2$. (b) Collision of a path can be detected by checking the condition on all sampled coordinates.

This type of map efficiently detects collisions with the Bézier curve parametric path. To describe the method, some notation must be introduced. Given coordinate $\mathbf{B}(t_j)$, γ_j denotes the transformed distance used to construct the bidirectional distance map, i.e.,

$$\gamma_j = \begin{cases} \|\mathbf{B}(t_j) - \mathbf{o}_j\|, & \text{if } \mathbf{B}(t_j) \in \mathcal{C}_{\text{free}} \\ -\|\mathbf{B}(t_j) - \mathbf{o}_j\|, & \text{if } \mathbf{B}(t_j) \in \mathcal{C}_{\text{obstacle}} \end{cases}. \tag{28}$$

When free space $\mathcal{C}_{\text{free}}$ contains the coordinate, $\mathbf{o}_j$ is defined to be the location of its nearest obstacle, and γ_j gives the distance between them. Otherwise, $\mathbf{o}_j$ is defined to be the location of the nearest free cell, and γ_j is the negatively signed distance between them. Differentiating (28) with respect to λ yields

$$\frac{\partial \gamma_j}{\partial \lambda} = \frac{\mathbf{B}(t_j) - \mathbf{o}_j}{\gamma_j} \cdot \frac{\partial \mathbf{B}(t_j)}{\partial \lambda}. \tag{29}$$

With the notation $\mathbf{p}_i$, $i = 0, 1, \ldots,$ is used to denote the end points of the global path segments, as shown in Fig. 6(a). Given the path segment connecting $\mathbf{p}_0$ and $\mathbf{p}_1$ if $\gamma(\mathbf{p}_1)$ is greater than $s_0 = \|\mathbf{p}_1 - \mathbf{p}_0\|$, then every point on line segment $\mathbf{p}_1 - \mathbf{p}_0$ is collision free against $\mathcal{C}_{\text{obstacle}}$. This is shown in Fig. 6(a). Moreover, it is likewise true for $\mathbf{p}_1$ and $\mathbf{p}_2$. As a result, the collision of an entire path can be detected by verifying this condition for $\mathbf{p}_i$, $\forall\, i$, which is shown in Fig. 6(b). In the following, the constrained global path optimization is formulated using γ on the bidirectional transformed distance map.

B. Constrained Optimization Formulation

If a global path $\mathcal{P}$ is composed of $m+1$ Bézier curves $\{\mathbf{B}^{[0]}(t), \ldots, \mathbf{B}^{[m]}(t)\}$, which is shown in Fig. 7, if the ith curve $\mathbf{B}^{[i]}(t)$ is given, and if $\mathbf{q}_i$, $\mathbf{q}_{i+1}$, and $\mathbf{h}_i$ denote the two end states and the control distance vector that represent the curve, then

$$\mathcal{P} = \bigcup_{i=0,\ldots,m} \mathbf{B}^{[i]}(t) = \bigcup_{i=0,\ldots,m} \mathbf{B}(\lambda_i, t) \tag{30}$$

where λ_i is the ith path parameter, i.e.,

$$\lambda_i = \begin{pmatrix} \mathbf{q}_i^T & \mathbf{h}_i^T & \mathbf{q}_{i+1}^T \end{pmatrix}^T, \quad i = 0, \ldots, m. \tag{31}$$

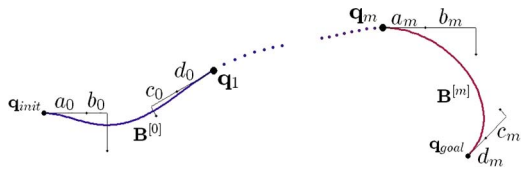


Fig. 7. Given respective initial and goal states $\mathbf{q}_{\text{init}}$ and $\mathbf{q}_{\text{goal}}$, respectively, the global path is a sequence of parametric Bézier curves $\mathbf{B}(\lambda_i, t)$, $i = 0, \ldots, m$.

It should be noted that the path's two end states must be fixed to the initial and goal states, i.e.,

$$\mathbf{q}_0 = \mathbf{q}_{\text{init}}, \quad \mathbf{q}_{m+1} = \mathbf{q}_{\text{goal}}$$

in which intermediate joint state $\mathbf{q}_i$, $i = 1, \ldots, m$, is commonly used by $\mathbf{B}^{[i-1]}(t)$ and $\mathbf{B}^{[i]}(t)$. Hence, the free variable vector that represents the global path is

$$\mathbf{X} = \begin{pmatrix} \mathbf{h}_0^T & \mathbf{q}_1^T & \mathbf{h}_1^T & , \cdots , & \mathbf{h}_{m-1}^T & \mathbf{q}_m^T & \mathbf{h}_m^T \end{pmatrix}^T. \quad (32)$$

This form is more compact than that of the naive control point representation $\mathbf{X} = \{\mathbf{B}_0^{[0]}, \ldots, \mathbf{B}_5^{[0]}, \ldots, \mathbf{B}_0^{[m]}, \ldots, \mathbf{B}_5^{[m]}\}$ because it reduces the parameter's dimension from $12(m+1)$ to $8m+4$. The path parameter can be optimized by solving the following NLP problem:

Minimize

$$J(\mathbf{X}) = \sum_{i=0}^{m} \sum_{j=0}^{k_i - 1} L^{[i]}(\lambda_i) = \sum_{i=0}^{m} \sum_{j=0}^{k_i - 1} \left[w_s \cdot s_j^{[i]} + w_\kappa \cdot \left(\kappa_j^{[i]}\right)^2 - w_\gamma \cdot \gamma_j^{[i]} \right] \quad (33)$$

subject to

$$\left(\kappa_j^{[i]}\right)^2 < \kappa_{\max}^2, \quad (34)$$

$$\gamma_j^{[i]} > s_{j-1}^{[i]}, \text{ and } \gamma_j^{[i]} > s_j^{[i]} \quad (35)$$

in which $L^{[i]}(\lambda_i)$ is the cost function of $\mathbf{B}^{[i]}$. The number of sample states k_i is proportional to the arc length of the ith curve. The first and second terms s and κ penalize arc length and curvature, respectively, and the curvature is constrained by (34) in a manner similar to the NLP discussed in Section II. The last term γ, which is defined by (28), measures safety in the form of collision avoidance. Equation (35) imposes its lower bounds on the lengths of joining segments, as shown in Fig. 6. Each term in (33) can be normalized by replacing the weighing factors with $\bar{w}_s = w_s \|\mathbf{p}_{\text{init}} - \mathbf{p}_{\text{goal}}\|$, $\bar{w}_\kappa = w_\kappa \kappa_{\max}^2$, and $\bar{w}_\gamma = w_\gamma \gamma_{\max}$, respectively, as follows:

$$L^{[i]}(\lambda_i) = \sum_{j=0}^{k_i - 1} \left[\bar{w}_s \frac{s_j^{[i]}}{\|\mathbf{p}_{\text{init}} - \mathbf{p}_{\text{goal}}\|} + \bar{w}_\kappa \frac{\left(\kappa_j^{[i]}\right)^2}{\kappa_{\max}^2} - \bar{w}_\gamma \frac{\gamma_j^{[i]}}{\gamma_{\max}} \right].$$

Again, the MMA algorithm is adapted to solve the optimization problem found in (33)–(35). Differentiating (33) yields

$$\frac{dL^{[i]}(\lambda_i)}{d\lambda_i} = \sum_{j=0}^{k_i - 1} \left[w_s \frac{\partial s_j^{[i]}}{\partial \lambda_i} + 2 w_\kappa \kappa_j^{[i]} \frac{\partial \kappa_j^{[i]}}{\partial \lambda_i} - w_\gamma \frac{\partial \gamma_j^{[i]}}{\partial \lambda_i} \right]$$

where $\partial s / \partial \lambda$, $\partial \kappa / \partial \lambda$, and $\partial \gamma / \partial \lambda$ are given by (26), (27), and (29), respectively. Stacking the gradients produces the gradient of the performance index:

$$\frac{dJ(\mathbf{X})}{d\mathbf{X}} = \begin{pmatrix} \frac{\partial L^{[0]}}{\partial \mathbf{h}_0} \\ \frac{\partial L^{[0]}}{\partial \mathbf{q}_1} + \frac{\partial L^{[1]}}{\partial \mathbf{q}_1} \\ \vdots \\ \frac{\partial L^{[m-1]}}{\partial \mathbf{q}_m} + \frac{\partial L^{[m]}}{\partial \mathbf{q}_m} \\ \frac{\partial L^{[m]}}{\partial \mathbf{h}_m} \end{pmatrix}.$$

It is important to note that $\mathbf{h}_i$ only affects $\mathbf{L}^{[i]}$, whereas $\mathbf{q}_i$ affects both $\mathbf{L}^{[i]}$ and $\mathbf{L}^{[i-1]}$ because it joins $\mathbf{B}^{[i]}$ and $\mathbf{B}^{[i-1]}$.

C. Merging

The solution found using the search graph (see Section III) is a good initial guess when solving the NLP defined by (33)–(35). It should be noted that both of the initial prediction's accuracy and the solution's compactness impacts optimality and the rate of convergence. To ensure compactness, consecutive Bézier curve primitives are merged.

The first step computes the radius-normalized minimum curvature control set offline. The set is obtained by solving the NLP in (21) and (22) with $w_s = 0$ for two end states, such that the length between them equals one. To describe the next step, $\mathbf{q}_{\tilde{c}-1}$, $\mathbf{q}_{\tilde{c}}$, and $\mathbf{q}_{\tilde{c}+1}$ will denote the series of end states connected by two adjacent primitives. The control set is translated and rotated to align with $\mathbf{q}_{\tilde{c}-1}$, which is shown in Fig. 8(a), and stretched using R, the Euclidean distance between $\mathbf{q}_{\tilde{c}-1}$, and $\mathbf{q}_{\tilde{c}+1}$, which are shown in Fig. 8(b). A search determines the element closest to the two primitives, which is designated by the yellow curve in Fig. 8(c). Here, $R(a^*, b^*, c^*, d^*)$ signifies the control length vector, and the two primitives are replaced with a single quintic Bézier curve, which is determined using $\mathbf{q}_{\tilde{c}-1}$, $R(a^*, b^*, c^*, d^*)$, and $\mathbf{q}_{\tilde{c}+1}$ if the curve avoids collision. This process is repeated in a bottom-up manner.

Fig. 9 shows the effectiveness of the merging process. Given an initial guess (dashed magenta curve), each image shows the optimal solution of the NLP (solid blue curve) obtained when curvatures and proximity to obstacles ($\bar{w}_s = 0$, $\bar{w}_\kappa = \bar{w}_\rho = 1$), are equally penalized, and the merging is processed with depths of 0, 1, and 6. Depth d specifies the maximum number of recursions necessary to merge at most 2^d adjacent primitives. The resulting curves have more degrees of freedom and deform as the size of $\mathbf{X}$ decreases due to merging. An attractive property of the parametric Bézier curve is that it sustains each primitive curve's smoothness against deformation. (The following section highlights this benefit when compared with the existing method presented in [3].) This means that an increase in the number of merging steps produces convergence to a better solution. In addition, a reduction in the size of $\mathbf{X}$ results in faster computation. In this example, the solution to Fig. 9(c) was computed approximately 10 times faster than the solution to Fig. 9(a).

V. Results

Here, empirical results for the proposed path planner are provided.

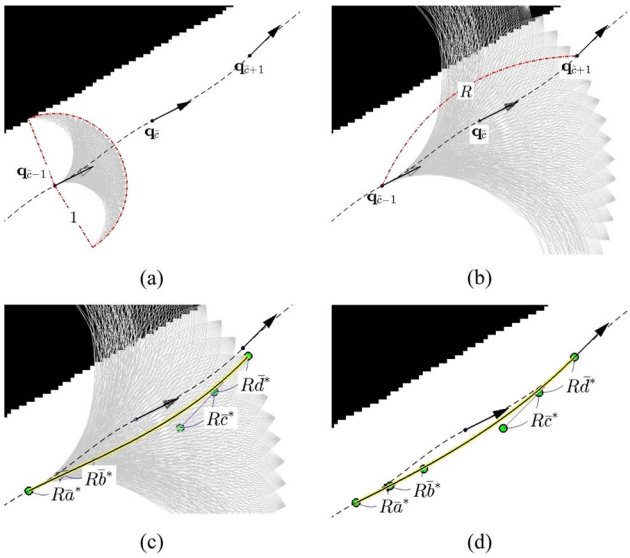


Fig. 8. Merging process on two consecutive path segments. (a) Translation and rotation of control set aligned with the first state. (b) Set stretched using the distance between end points. (c) Search for closet control that reaches the end state. (d) Replacing segments with the searched control. (a) Translation and rotation. (b) Scaling. (c) Lookup. (d) Merge.

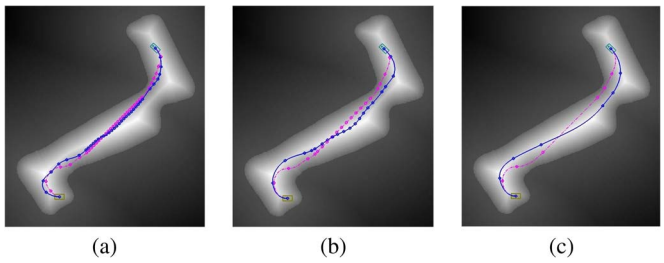


Fig. 9. Optimal solutions (solid blue curve) depend on the merge depth. Depth d specifies the maximum number of recursions necessary to achieve the merging of at most 2^d adjacent primitives. Joining states that result from merging are visible through void circles. (a) No *merge*. (b) Depth $= 1$. (c) Depth $= 6$.

Fig. 10. (a) TUT mobile laboratory area. (b) Test field encoded into a 2-D occupancy grid map with resolution of 0.2 m using a 2-D laser servo [27].

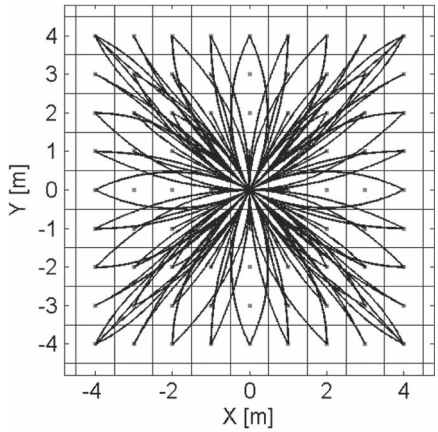


Fig. 11. State lattice control set used in the experiments.

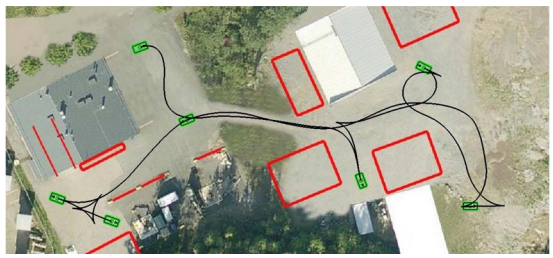


Fig. 12. Solutions to proposed path planner for multiple waypoints in the TUT mobile laboratory area.

A. Field Experiments

The proposed path-planning algorithm was successfully implemented on our autonomous wheeled loader, i.e., a mobile GIM [see Fig. 1(b)], operating in the TUT mobile laboratory area. The GIM uses a *Hurricane-QM57* computer with an *Intel Core i7* processor and 8 GB of RAM. The robot gathers laser and navigation data using a tilting 2-D laser-servo *SICK LMS111* and localizes data using an IMU-enabled GPS with wheel odometry. Using sensors, the test field was encoded offline into a 2-D occupancy grid map with dimensions of 150 m by 80 m and resolution 0.2 m, which is shown in Fig. 10, to preprogram the global scale geographic information, such as buildings, trees, and vegetation, while ensuring that local obstacles, such as traffic cones and trash bins, were detected online. More detail of this localization and mapping algorithm can be found in [27].

The state lattice builder discretized the state space using a 1-m position resolution and 16 discrete headings: 0, $\tan^{-1}(1/2)$, $\pi/4$, and their symmetries by rotation and reflection. Fig. 11 shows the kinematic constrained control set used in the experiments. The elements' minimum turning radius was designed to be greater than two cells to satisfy the robot's maximum curvature of 0.5^{-1} m. Paths determined by the lattice planner produced errors in the end states due to this discretization. However, the path optimization procedure compensated for these errors to precisely perform the desired task.

Fig. 12 shows the planned paths for multiple waypoints using the proposed algorithm. Given a planned path, the control commands are computed using previous work [4].

Fig. 13 exhibits snapshots of the reactive path planning, which avoids dynamically detected obstacles using a laser sensor. The robot initially plans the paths with the precomputed global map and replans using incrementally updated laser data.

The real-time reactive path planning algorithm is implemented, and the GIM robot successfully operates in the field, as shown in Fig. 14. Path computation was sufficiently fast to be applied to the online replanning. Typical run times involving an A^* search in the state lattice and optimization were less than 100 ms.

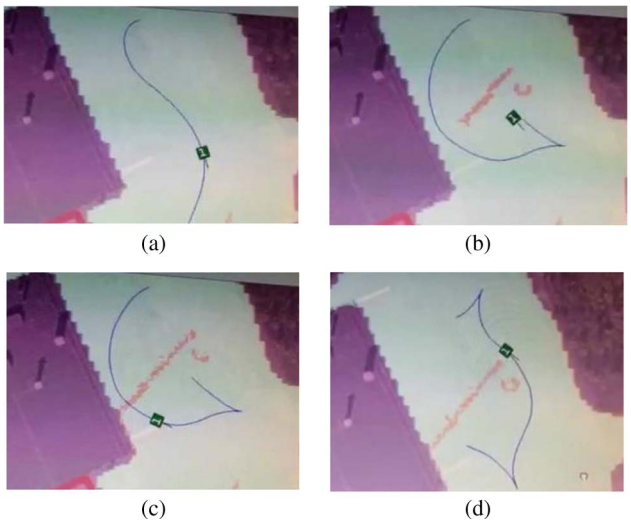


Fig. 13. Reactive path planning designed to avoid dynamically detected obstacles. (a) Given prior global environment knowledge, an initial path (blue curve) is planned. (b) If the laser sensor detects obstacle cells intersecting the path as the robot follows it, the path is replanned. (c) and (d) This process is repeatedly performed until the robot reaches the goal state.

Fig. 14. Autonomous driving test in which robot reactively navigates dynamically changing obstacles.

B. Performance

The performance of the proposed planner was demonstrated and compared with the work of Dolgov *et al.* presented in [3]. The primary difference between the two methods is the parametric path representation. As in (32), the proposed method uses joint states and control distance vectors to represent the path as a piecewise Bézier curve. In contrast, Dolgov *et al.* employed discrete coordinates to represent a piecewise linear path. Henceforth, the proposed Bézier curve parametric path optimization method will be denoted BPO, and the coordinates parametric path optimization method presented in [3] will be denoted CPO.

Fig. 15 shows the optimal solution derived using the CPO method. This method performs a two-stage optimization procedure. In the first stage, the NLP is formulated as in (33)–(35) using the coordinates of the vertices to parameterize a path. In the second stage, this method interpolates the first-stage solution by performing another optimization and using higher resolution path discretization (small dots) while maintaining the original vertices (void circles).

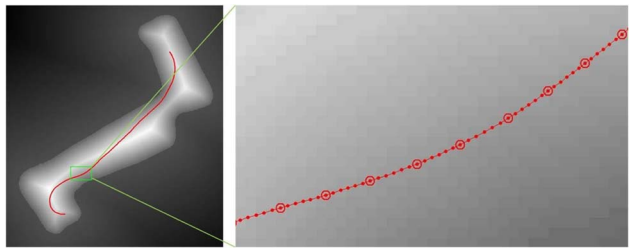


Fig. 15. (Left) Optimized path produced using CPO method. (Right) Path constructed using discrete vertices optimized in two stages. Void circles and small dots indicate the vertices computed in the first and second stages.

Fig. 16 shows the effectiveness of the BPO method when compared with the CPO method. Given identical initial guesses (dashed magenta curve), the figure compares solutions to the NLP using the CPO (solid red curve) and BPO (solid blue curve) methods, depending on the size of path parameter $\mathbf{X}$ (in each column). The initial guess is a sequence of Bézier curves found in the state lattice. When applying the BPO method, the size of $\mathbf{X}$ decreases, when merging with depths of 0, 1, 3, and 6, from left to right. Each column obtained using the CPO method samples equally spaced vertices on the path so that the size of $\mathbf{X}$ equals that of the BPO method. To highlight the differences between the two solutions, an identical NLP (33)–(35) was used, which equally penalizes the curvature and proximity to obstacles: $\bar{w}_s = 0$, $\bar{w}_\kappa = \bar{w}_\rho = 1$.

In general, the resulting curves have more degrees of freedom and deform as the size of $\mathbf{X}$ decreases. The CPO method reveals its drawback to be the generation of sharp corners as the path vertices grow too coarse. These corners can be smoothed by performing multiple-stage interpolations, but this will compound the computational complexity. Conversely, the BPO method's paths converge to better solutions, while maintaining each primitive curve's smoothness, as the size of $\mathbf{X}$ decreases due to merging. Consequently, the BPO method's solution produces smoother paths than those produced by the CPO method and better approximates the Voronoi diagram path.

Fig. 17 more clearly demonstrates the superiority of the BPO method when applied in a larger more complex workspace. Whereas the CPO method's solution (dashed red curve) barely transforms from the initial guess (dashed magenta curve), the BPO method's solution (solid blue curve) converges to a much better optimum. The zoomed-in top-left image highlights another benefit of the BPO method when applied to cusp-point optimization. The cusp point indicates the point at which the robot switches its direction. The CPO method divides a path into forward and backward segments and then anchors the points as boundary conditions and independently optimizes each one. This approach can cause a drastic increase in the required amount of computation as the number of the points increases. On the other hand, the BPO method has degrees of freedom, which deform at the point by parameterizing it as a joint state between two different directional Bézier curves, and all segments can be simultaneously optimized. The resulting

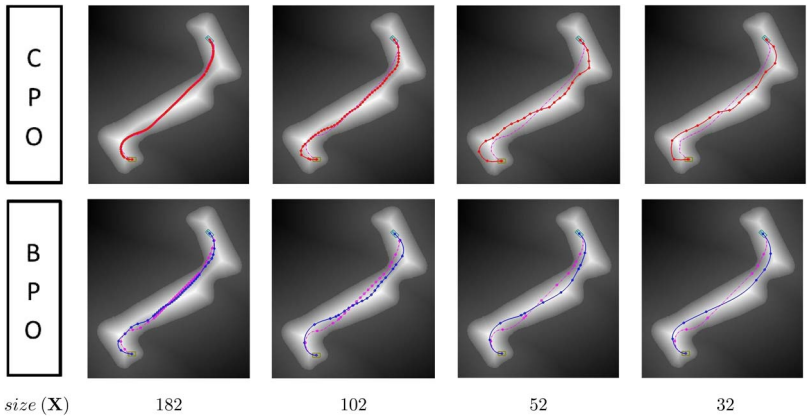


Fig. 16. Given initially predicted path (dashed magenta curve), optimal solutions obtained using the CPO (red solid curves in the first row) and BPO (solid blue curve in the second row) methods, depending on the size of path parameter $\mathbf{X}$ (in each column).

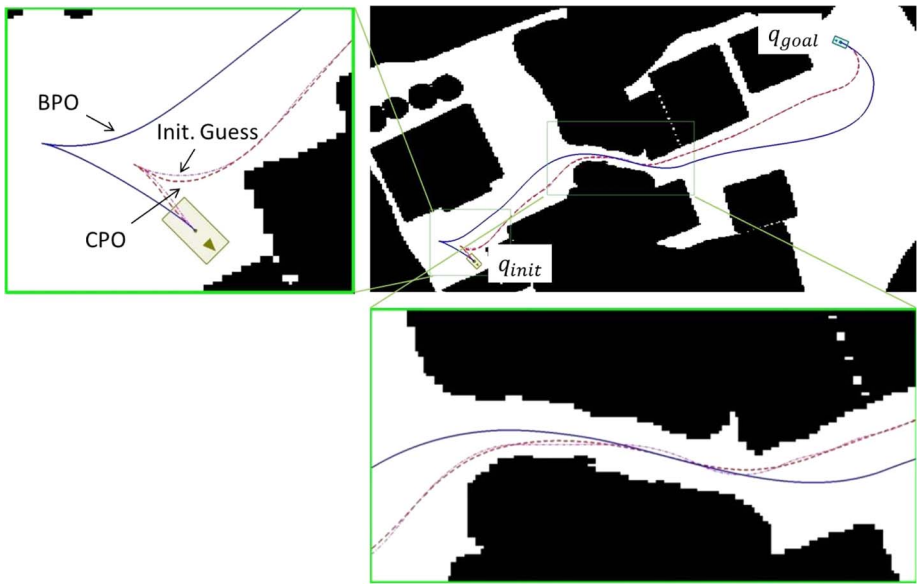


Fig. 17. Comparison of global path optimization using CPO (dashed red curve) and BPO (solid blue curve) methods. (Top-left) BPO method generates smoother and safer paths than the CPO method around a cusp point. (Bottom) BPO method's solution achieves a smoother and safer trajectory than the CPO method in a narrow corridor.

solution is not only calculated faster, but it also achieves better optimality. In addition, the bottom-right image clearly shows that the BPO method outperforms the CPO method as the robot navigates a narrow corridor, in which planning a smooth path that approaches the Voronoi diagram is crucial.

Fig. 18 provides a numerical comparison of the planning performances for 4000 randomly generated test cases in an identical work environment. Both methods were implemented using identical programming infrastructures and NLP parameters. The initially predicted paths were searched in the state lattice. The CPO method sampled equally spaced vertices on the paths. As shown in Fig. 16, the sampling rates that match the size of $\mathbf{X}$ to that of the BPO method may generate sharp corners and violate the maximum curvature constraint. In this simulation, it results in a 56% violation rate. Therefore, a sampling interval of 1 and 0.2 m were used for the first and second optimization stages, respectively, as was suggested in [3]. The BPO method (blue circles) outperformed the CPO method (red triangles) in every area: run time [see Fig. 18(a)], magnitude of curvature [see Fig. 18(b)], and distances to nearest obstacle [see Fig. 18(c)]. While the CPO method required an average of 3.28 s to compute a path, the BPO method needed only 300 ms. These magnitudes, as they relate to curvature and the distance to nearest obstacles, increase by factors greater than 8.9 and 1.295, respectively.

VI. Conclusion

This paper has proposed a new efficient path-planning algorithm. The path planner consists of offline and online subsystems. The offline planner precomputes feasible motion primitives based on single Bézier curves by solving two point boundary value problems. The resulting motion primitives set is input into the online planner to convert the constrained motion planning into a graph search. The shortest path, which is determined by applying the A^* algorithm through the search graph, is composed of a sequence of quintic Bézier curves. The last step improves the solution's quality by solving the

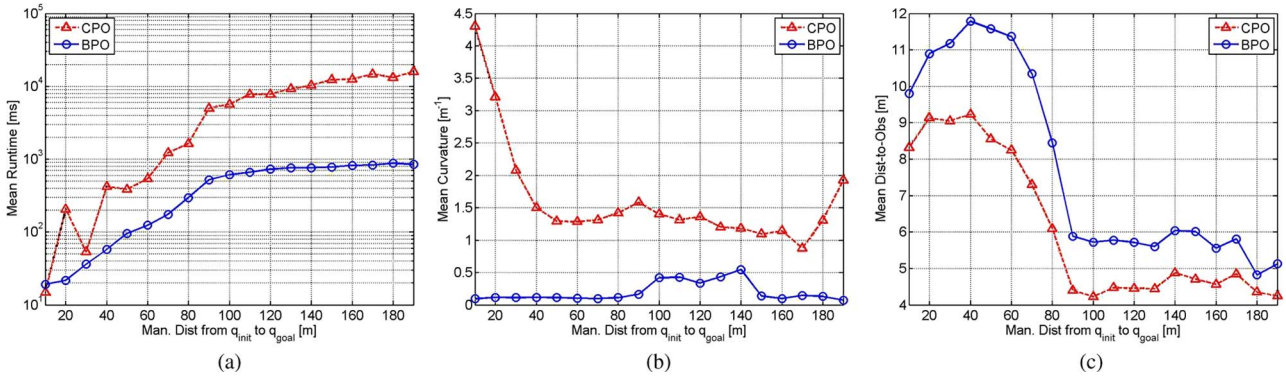


Fig. 18. Performance comparison between CPO and BPO methods, depending on distance maintained between initial and goal states. (a) Mean run time. (b) Mean curvature. (c) Mean distances to obstacles.

constrained path optimization problem. Numerical simulations demonstrate the proposed algorithm's significant path-optimization improvement over the existing algorithm [3]. In addition, experimental results demonstrate not only successful generation of safe routes but comfortable control results for autonomous ground vehicles as well, which operate in semi-structured environments in real time.

REFERENCES

[1] M. Oetiker, G. Baker, and L. Guzzella, "A navigation-field-based semi-autonomous nonholonomic vehicle-parking assistant," *IEEE Trans. Veh. Technol.*, vol. 58, no. 3, pp. 1106–1118, Mar. 2009.

[2] M. Likhachev and D. Ferguson, "Planning long dynamically-feasible maneuvers for autonomous vehicles," *Int. J. Robot. Res.*, vol. 28, no. 8, pp. 933–945, Aug. 2009.

[3] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *Int. J. Robot. Res.*, vol. 29, pp. 485–501, 2010.

[4] R. Ghabcheloo *et al.*, "Autonomous motion control of a wheel loader," in *Proc. ASME Dyn. Syst. Control Conf.*, 2009, pp. 427–434.

[5] T. Maekawa, T. Noda, S. Tamura, T. Ozaki, and K. Machida, "Curvature continuous path generation for autonomous vehicle using b-spline curves," *Comput.-Aided Des.*, vol. 42, no. 4, pp. 350–359, 2010.

[6] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: The field d^* algorithm," *J. Field Robot.*, vol. 23, no. 2, pp. 79–101, Feb. 2006.

[7] T. Howard, C. Green, D. Ferguson, and A. Kelly, "State space sampling of feasible motions for high-performance mobile robot navigation in complex environments," *J. Field Robot.*, vol. 25, no. 6/7, pp. 325–345, Jun. 2008.

[8] S. Koenig and M. Likhachev, "D* lite," in *Proc. AAAI*, 2002, pp. 476–483.

[9] A. Nash, K. Daniel, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," in *Proc. Nat. Conf. Artif. Intell.*, 2007, pp. 1177–1183.

[10] L. E. Dubins, "On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents," *Amer. J. Math.*, vol. 79, no. 3, pp. 497–516, Jul. 1957.

[11] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific J. Math.*, vol. 145, no. 2, pp. 367–393, 1990.

[12] T. Fraichard, "Smooth trajectory planning for a car in a structured world," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1991, pp. 318–323.

[13] Y. Kanayama and N. Miyake, "Trajectory generation for mobile robots," *Robot. Res.*, vol. 3, pp. 333–340, 1986.

[14] F. Lamiraux and J.-P. Laumond, "Smooth motion planning for car-like vehicles," *Amer. J. Math.*, vol. 17, no. 4, pp. 498–502, 2001.

[15] T. Fraichard and A. Scheuer, "From reeds and shepp's to continuous-curvature paths," *IEEE Trans. Robot.*, vol. 20, no. 6, pp. 1025–1035, Dec. 2004.

[16] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *Proc. IEEE/RSJ Int. Conf. IROS*, 2009, pp. 2427–2433.

[17] J.-W. Choi, R. E. Curry, and G. H. Elkaim, "Continuous curvature path generation based on bezier curves for autonomous vehicles," *IAENG Int. J. Appl. Math.*, vol. 40, no. 2, pp. 91–101, May 2010.

[18] S. Fallah, B. Yue, O. Vahid-Araghi, and A. Khajepour, "Energy management of planetary rovers using a fast feature-based path planning and hardware-in-the-loop experiments," *IEEE Trans. Veh. Technol.*, vol. 62, no. 6, pp. 2389–2401, Jul. 2013.

[19] I. Dolinskaya, "Time-optimal trajectories with bounded curvature in anisotropic media," *Int. J. Robot. Res.*, vol. 31, pp. 1761–1793, 2012.

[20] H. Chitsaz, "On time-optimal trajectories for a car-like robot with one trailer," in *Proc. Conf. Control Appl.*, 2013, pp. 114–120.

[21] A. Kelly and B. Nagy, "Reactive nonholonomic trajectory generation via parametric optimal control," *Int. J. Robot. Res.*, vol. 22, no. 7–8, pp. 583–601, Jul. 2003.

[22] D. Ferguson, T. Howard, and M. Likhachev, "Motion planning in urban environments: Part I," in *Proc. IEEE/RSJ Int. Conf. IROS*, 2008, pp. 1063–1069.

[23] K. Svanberg, "A class of globally convergent optimization methods based on conservative convex separable approximations," *SIAM J. Optim.*, pp. 555–573, 2002.

[24] M. Pivtoraiko and A. Kelly, "Generating near minimal spanning control sets for constrained motion planning in discrete state spaces," in *Proc. IEEE/RSJ Int. Conf. IROS*, Aug., 2005, pp. 3231–3237.

[25] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 90–98, 1986.

[26] S. Beriault, F. Subaie, K. Mok, A. Sadikot, and G. Pike, "Automatic trajectory planning of dbs neurosurgery from multi-modal mri datasets," in *Proc. Med. Image Comput. Comput.-Assisted Intervention*, 2011, vol. 6891, pp. 259–266.

[27] A. Kolu, M. Lauri, M. Hyvonen, and R. Ghabcheloo, "A mapping method tolerant to calibration and localization errors based on tilting 2d laser scanner," presented at the Eur. Control Conf., Linz, Austria, 2015, Paper ThC11.2.

Ji-wung Choi (M'11) was born in Korea. He received the B.S. degree in electrical engineering from Yonsei University, Seoul, Korea, in 2006 and the Ph.D. degree in computer engineering from the University of California, Santa Cruz, CA, USA, in 2010.

He is currently a Principal Research Engineer with Hyundai MOBIS Company, Ltd., Uiwang, Korea. His research interests include motion planning and optimal control of autonomous robots.

Kalevi Huhtala was born in August 1957. He received the Dr. Tech. degree from Tampere University of Technology, Tampere, Finland, in 1996.

He is currently a Professor with the Department of Intelligent Hydraulics and Automation, Tampere University of Technology, where he is also Head of the department. His primary research interests include intelligent mobile machines and diesel engine hydraulics.