2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA

Constrained Path Optimization with Bézier Curve Primitives

Ji-Wung Choi and Kalevi Huhtala

Abstract— **This article suggests a novel path planning algorithm for a non-holonomic wheeled vehicle operating in a semi-structured environment. The first step of the algorithm is to compute offline a finite set of feasible motions connecting discrete robot states to construct a search graph. The motion primitives based on Bézier curves are generated by solving the constrained optimization problem (*COP*). Applying A^* on the search graph produces paths in the form of a sequence of the primitives. Although the sequence is drivable and suboptimal, we perform online path smoothing to remove the jerky or extraneous motions. The post-procedure is done by using a gradient based method to solve another *COP*. The numerical simulations show remarkable performance improvement in the constrained path optimization by applying our algorithm, compared to other existing works. Also, field experimental results demonstrate successful generation of fast and safe trajectories for real-time autonomous driving.**

I. Introduction

This paper addresses a path planning problem for a non-holonomic wheeled vehicle moving on a plane. Our work on this problem is motivated by the application of autonomous wheeled loaders carrying pallets in ports, construction sites, factories, and so on. To pick up the pallet, the robot must reach precisely in front of and aligned with the fork holes. Also, the resulting path must satisfy robotic system's constraints such as non-holonomic nature and the maximum curvature. This paper presents a computationally efficient solution to the path planning problem. The resulting path is generated by joining quintic Bézier curves in a smooth way from A^* search through discrete state spaces and then smooth the sequence of curves by solving the constrained optimization problem using the globally-convergent method-of-moving-asymptotes (MMA) algorithm [1].

Many work on A^* based motion queries gain computational efficiency in $2D$ grids [2] and [3]. However, discrete transitions connecting the grid cells produce sharp corners which is infeasible for non-holonomic robots to follow. Even though the drawback can be overcome by adapting higher resolution local planner that accounts for robot mobility constraints, there still remains a possibility of collision between the robot and obstacles in cluttered environments. This gives rise to necessity of some path smoothing methods. Quintic Bézier curves have been widely used to produce curvature-continuous paths by smoothing straight-line paths [4] and [5]. Lau et al. used the curve connecting two consecutive waypoints to generate time-optimal trajectories under kinodynamic constraints of a mobile robot [4]. Instead, Choi et al. added the curve on the corner area around a waypoint [5]. Their numerical simulations demonstrate superiority of the approach to [4], in terms of smoothness. While the work of [5] is limited to structured environments such as highways and racecourses, this paper extends the work to unstructured and more cluttered environments.

Pivtoraiko and Kelly [6] proposed the *state lattice*, a graph constructed by discretized states and feasible motion segments, to keep a balance between the computational advantages and feasibility. The approach has been implemented on motion planning for driverless cars [7], where the motion segments are represented by cubic polynomial curvature functions of path length and obtained by solving the constrained optimization problem [8]. In the set of works, global path smoothing has not been considered. Instead, a local trajectory has been repeatedly generated to converge onto the global trajectory. Although such paths are drivable and suboptimal, they can contain unnatural swerves that require unnecessary steering, especially in a complex environment. For this reason, Dolgov et al. [9] refined the global path by using the conjugate gradient method. They parameterized the path based on the coordinates of the vertices. The parametric path is simple to implement, but requires extra step to interpolate the optimized coordinates into smooth curves.

Main contribution of this paper is the local and global optimization of the planned path using few parameters, which efficiently solves the problem of constrained path planning. Our approach is novel in terms of a path regularization procedure that merges consecutive segments of a path by substituting them with an approximating longer segment, as presented in Sec. IV-B. We demonstrate remarkable performance improvement in path optimization by applying our algorithm, compared to [9], in Sec. V.

II. Bézier Curve Parametric Path

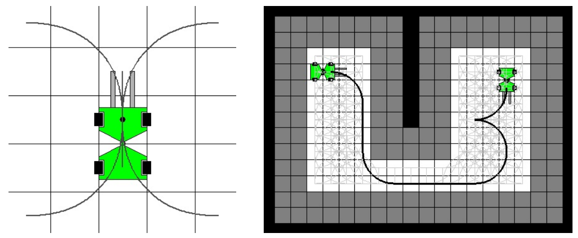


Fig. 1. Left: The lattice is composed of discretized states. A finite set of feasible motion segments is designed to connect between the states. Right: A search graph is constructed by copying the set to a free space states.

The first step of path planning is to convert a $2D$ workspace into the *state lattice* [6], a graph representation

J.-W. Choi and K. Huhtala are with Department of Intelligent Hydraulics and Automation, Tampere University of Technology, 33101 Tampere, Finland {ji.choi,kalevi.huhtala}@tut.fi

of a discretized configuration space, where repeating pairs of configurations are connected by a finite set of feasible motion segments. Fig. 1 illustrates an example. Six solid curves in the left image indicate the motion segments. The search graph is constructed by copying the segments at states in a free space, $\mathcal{C}_{free}$. The space is a complementary set of $\mathcal{C}_{obstacle}$ generated by inflating all obstacle cells by the outer radius of the robot. (In the right image, black and gray cells indicate obstacles and inflated space, respectively.) Then, A^* is applied on the graph to produce the shortest path (black solid curve in the right image).

In this paper, Bézier curves are used to generate the motion segments. A Bézier Curve of degree n, $\mathbf{B}(t)$ is defined by $n+1$ control points $\mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_n$:

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

We decided to use quintic Bézier curves. This is because the basic requirement of path planning is to pass through the beginning and end positions (x,y), with specified orientations θ and curvatures κ. The least degree of the Bézier curve that can satisfy the requirement is five as explained below.

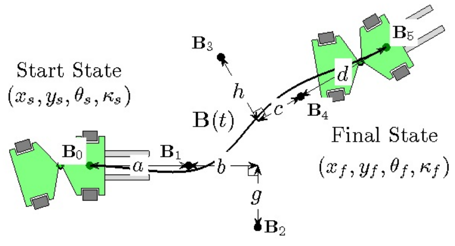


Fig. 2. A quintic Bézier curve $\mathbf{B}(t)$ can be determined by 12-tuple $\lambda = \{x_s, y_s, \theta_s, \kappa_s, a, b, c, d, x_f, y_f, \theta_f, \kappa_f\}$, where $(x_s, y_s, \theta_s, \kappa_s)$ is the start state, $(x_f, y_f, \theta_f, \kappa_f)$ is the final state, and (a,b,c,d) is control distance vector.

Suppose that we are to generate a motion segment, in the form of a quintic Bézier curve, connecting two states $\mathbf{q}_s = (x_s, y_s, \theta_s, \kappa_s)^T$ and $\mathbf{q}_f = (x_f, y_f, \theta_f, \kappa_f)^T$, as illustrated in Fig. 2. Since Bézier Curves start at $\mathbf{B}_0$ and stop at $\mathbf{B}_n$, the two end control points must be set to the two end positions:

$$\mathbf{B}_0 = \mathbf{p}_s = (x_s, y_s)^T, \quad \mathbf{B}_5 = \mathbf{p}_f = (x_f, y_f)^T. \tag{2}$$

Since Bézier Curves are tangent to the lines connecting the first (and last) two control points, $\mathbf{B}_1$ and $\mathbf{B}_4$ are

$$\mathbf{B}_1 = \mathbf{p}_s + a\hat{\theta}_s, \quad \mathbf{B}_4 = \mathbf{p}_f - d\hat{\theta}_f, \quad a, d \in \mathbb{R}, \tag{3}$$

where $\hat{\theta}$ is the unit vector corresponding to θ. The curvatures of Bézier Curves at end endpoints are given by

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

where g is the perpendicular distance from $\mathbf{B}_2$ to $\mathbf{B}_1 - \mathbf{B}_0$, and h is similarly defined backward. (See Fig. 2.) The equations above constrain $\mathbf{B}_2$ and $\mathbf{B}_3$ to be

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

As the result, the curve is spanned by the 12-tuple parameter $\lambda = \begin{pmatrix} x_s & y_s & \theta_s & \kappa_s & a & b & c & d & x_f & y_f & \theta_f & \kappa_f \end{pmatrix}^T$.

Incorporating (2)-(4) into (1) has the curve represented as

$$\mathbf{B}(\lambda, t) = \mathbf{F}(\lambda)\mathbf{G}(t), \tag{5}$$

where

$$\mathbf{F}(\lambda) = \begin{pmatrix} \mathbf{p}_s & a^2\kappa_s\dot{\hat{\theta}}_s & a\hat{\theta}_s & b\hat{\theta}_s & c\hat{\theta}_f & d\hat{\theta}_f & d^2\kappa_f\dot{\hat{\theta}}_f & \mathbf{p}_f \end{pmatrix}, \tag{6}$$

$$\mathbf{G}(t) = \begin{pmatrix} (1-t)^3(6t^2+3t+1) \\ \frac{25}{2}t^2(1-t)^3 \\ 5t(1-t)^3(t+1) \\ 10t^2(1-t)^3 \\ -10t^3(1-t)^2 \\ 5t^3(1-t)(t-2) \\ \frac{25}{2}t^3(1-t)^2 \\ t^3(6t^2-15t+10) \end{pmatrix}. \tag{7}$$

The form of (5) has the derivatives of $\mathbf{B}(t)$ represented as

$$\dot{\mathbf{B}}(t) = \mathbf{F}(\lambda)\dot{\mathbf{G}}(t). \tag{8}$$

Denoting $\mathbf{F}_1(\lambda)$ and $\mathbf{F}_2(\lambda)$, the first and second row vector of $\mathbf{F}(\lambda)$, the partial derivatives of $\mathbf{B}$ and $\dot{\mathbf{B}}$ are given by

$$\begin{aligned} \frac{\partial \mathbf{B}_x(t)}{\partial \lambda} &= \dot{\mathbf{F}}_1(\lambda)\mathbf{G}(t), & \frac{\partial \mathbf{B}_y(t)}{\partial \lambda} &= \dot{\mathbf{F}}_2(\lambda)\mathbf{G}(t), \\ \frac{\partial \dot{\mathbf{B}}_x(t)}{\partial \lambda} &= \dot{\mathbf{F}}_1(\lambda)\dot{\mathbf{G}}(t), & \frac{\partial \dot{\mathbf{B}}_y(t)}{\partial \lambda} &= \dot{\mathbf{F}}_2(\lambda)\dot{\mathbf{G}}(t). \end{aligned}$$

Since dimension of λ is 12, equal to that of $\{\mathbf{B}_0, \ldots, \mathbf{B}_5\}$, the formulations have no advantage when representing a single curve. However, it is very efficient to parameterize a piecewise Bézier curves path and hence leads to improvement of the optimality and computation speed in the path optimization, as discussed in Sec. IV.

Note that the parametric Bézier curve starting from $\mathbf{q}_s$ and reaching to $\mathbf{q}_f$ has four degrees of freedom spanned by $\mathbf{h} = (a,b,c,d)^T$. We are to find the optimal control distance vector $\mathbf{h}^*$ that generates smooth paths under the maximum curvature constraint of the robot. It can be formulated into the following non-linear programming (*NLP*) problem.

$$\text{Minimize: } J(\mathbf{h}) = \sum_{j=0}^{k-1} \Big[w_s \cdot s_j(\mathbf{h}) + w_\kappa \cdot \kappa_j^2(\mathbf{h}) \Big], \tag{9}$$

$$\text{subject to: } \kappa_j^2 < \kappa_{max}^2. \tag{10}$$

The performance index J leads to short and smooth paths by penalizing discrete path lengths s_j and curvatures κ_j, associated with $\{t_0, \ldots, t_k\} \subset [0,1]$ such that

$$s_j = \|\mathbf{B}(t_{j+1}) - \mathbf{B}(t_j)\|, \tag{11}$$

$$\kappa_j = \mathrm{sgn}\Big(\dot{\mathbf{B}}(t_j) \times \dot{\mathbf{B}}(t_{j+1})\Big) \frac{\Delta\theta_j}{s_j}, \tag{12}$$

where $\Delta\theta_j$ is the change of orientation between t_j and t_{j+1}:

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

Although $\kappa(t)$, the curvature of a Bézier curve is a function of t, we use the discrete curvature (12) determined by two consecutive sample postures. This is because the function with discrete t_j can lead to singularity in which smoothness is overestimated when the curve has high curvature points between sample postures.

The solution with its maximum magnitude of curvature less than the robot's limit κ_{max} in (10) can be selected as a feasible motion segment.

We adapt the globally-convergent method-of-moving-asymptotes (MMA) algorithm [1] to solve the constrained optimization problem (9)-(10). It is widely known that the performance of gradient based methods, including MMA, heavily depends on two factors: (i) proximity of the initial guess to the optima and (ii) computational efficiency of the gradient of the performance index.

Empirically, we found a quarter length between $\mathbf{p}_s$ and $\mathbf{p}_f$ to be good initial guess for each control distance:

$$\mathbf{h}_{guess} = \|\mathbf{p}_f - \mathbf{p}_s\| \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{pmatrix}^T. \quad (13)$$

The gradient of the performance index (9) is determined by the partial derivative of s_j and κ_j:

$$\frac{dJ(\mathbf{h})}{d\mathbf{h}} = \sum_{j=0}^{k-1} \left[w_s \cdot \frac{\partial s_j}{\partial \mathbf{h}} + w_\kappa \cdot 2\kappa_j \frac{\partial \kappa_j}{\partial \mathbf{h}} \right]. \quad (14)$$

The derivatives are obtained by differentiating (11) and (12) with respect to λ:

$$\frac{\partial s_j}{\partial \lambda} = \frac{\mathbf{B}(t_{j+1}) - \mathbf{B}(t_j)}{\|\mathbf{B}(t_{j+1}) - \mathbf{B}(t_j)\|} \cdot \left(\frac{\partial \mathbf{B}(t_{j+1})}{\partial \lambda} - \frac{\partial \mathbf{B}(t_j)}{\partial \lambda} \right). \quad (15)$$

$$\frac{\partial \kappa_j}{\partial \lambda} = \frac{1}{s_j} \left[\frac{\partial \theta_{j+1}}{\partial \lambda} - \frac{\partial \theta_j}{\partial \lambda} - \kappa_j \frac{\partial s_j}{\partial \lambda} \right], \quad (16)$$

$$\frac{\partial \theta_j}{\partial \lambda} = \frac{\dot{\mathbf{B}}(t_j)}{\dot{\mathbf{B}}(t_j) \cdot \dot{\mathbf{B}}(t_j)} \times \frac{\partial \dot{\mathbf{B}}(t_j)}{\partial \lambda}, \quad (17)$$

III. Collision Avoidance

While obstacles do not need to be considered in motion primitive generation (9)-(10), the global path optimization must take care of collision avoidance. For effective implementation of collision avoidance, we use the *distance map*, the map where each discrete grid is labeled with the distance to the nearest obstacle. Typical distance map is constructed by computing the Euclidean distance transform of the binary image of workspace [9]. However, the map may not work well for reactive path optimization, especially when the initially planned path can be made in collision by obstacles detected while following the path. New search through the updated space can provide safe path to be optimized but sometimes causes computational delay that requires the robot to stop. To remedy the drawback, we adapt the *signed distance map* [10].

Fig. 3 compares the two different distance maps for an identical initial guess (left image). Typical distance map transforms the distance from obstacle cells to free cells. The middle image shows the potential field generated by negating the distance indicated by darkness. Since the distance remains constant in $\mathcal{C}_{obstacle}$ (black), an initial guess

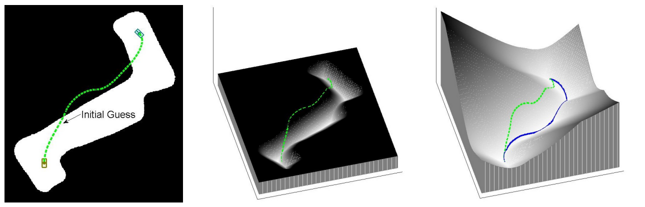


Fig. 3. Left: An initial guess path (dashed green) colliding obstacle. Middle: Adjusting the path in the potential field of typical distance map fails to find any collision-free solution. Right: The potential of the signed distance map pulls down the path into the optimal solution (blue solid).

colliding $\mathcal{C}_{obstacle}$ (green dashed) can be hardly adjusted into any collision-free solution. On the other hand, the signed distance map transforms the distance from/to obstacle cells to/from free cells with different signed values. We can see that the slope of the potential in obstacle space of the right image tends to pull down the initial guess path into the optimal solution (blue solid).

Given a coordinate $\mathbf{B}(t_j)$, let γ_j be the signed distance:

$$\gamma_j = \begin{cases} \|\mathbf{B}(t_j) - \mathbf{o}_j\|, & \text{if } \mathbf{B}(t_j) \in \mathcal{C}_{free}, \\ -\|\mathbf{B}(t_j) - \mathbf{o}_j\|, & \text{if } \mathbf{B}(t_j) \in \mathcal{C}_{obstacle}, \end{cases} \quad (18)$$

where $\mathbf{o}_j$ is defined as the location of the nearest obstacle when the coordinate is in $\mathcal{C}_{free}$, or the nearest free cell otherwise. Differentiating (18) yields

$$\frac{\partial \gamma_j}{\partial \lambda} = \frac{\mathbf{B}(t_j) - \mathbf{o}_j}{\gamma_j} \cdot \frac{\partial \mathbf{B}(t_j)}{\partial \lambda}. \quad (19)$$

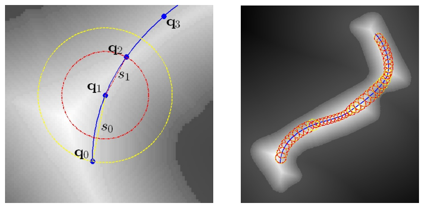


Fig. 4. Collision detection in the signed distance map.

This type of map is efficiently used to detect collision of the Bézier curve parametric path. As Fig. 4(a) illustrates, if $\gamma(\mathbf{q}_j)$ is greater than $\|\mathbf{q}_j - \mathbf{q}_{j-1}\|$, then every point on the line segment of $\mathbf{q}_j - \mathbf{q}_{j-1}$ is collision free against $\mathcal{C}_{obstacle}$. It also holds true between $\mathbf{q}_j$ and $\mathbf{q}_{j+1}$. As the result, collision of the whole path can be detected by checking the condition on all the sampled coordinates, as shown in Fig. 4(b). The collision checking is implemented at the constrained global path optimization (25) in the following section.

IV. Global Path Optimization

Recall that applying A^* on the state lattice (Fig. 1) generates the shortest path composed of a sequence of motion segments. Since each segment is created by solving the constrained optimization problem (9)-(10), the resulting paths are not only feasible but sub-optimal motions to drive. However, it can provide unnatural swerves in a complex environment [9]. This motivated us to further optimize the sequence of curves.

A. Constrained Optimization Formulation

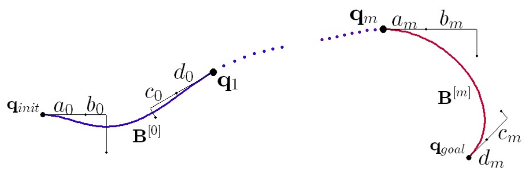


Fig. 5. Given the initial and goal states, $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, the global path is a sequence of parametric Bézier curves $\mathbf{B}(\lambda_i,t)$, $i=0,\ldots,m$.

Suppose that a global path $\mathcal{P}$ is composed of $m+1$ Bézier curves $\{\mathbf{B}^{[0]}(t),\ldots,\mathbf{B}^{[m]}(t)\}$ as shown in Fig. 5. Given i-th curve $\mathbf{B}^{[i]}(t)$, we let $\mathbf{q}_i$, $\mathbf{q}_{i+1}$ and $\mathbf{h}_i$ denote the two end states and control distance vector to determine the curve. That is,

$$\mathcal{P} = \mathbf{B}^{[0]}(t)\cup\cdots\cup\mathbf{B}^{[m]}(t) = \mathbf{B}(\lambda_0,t)\cup\cdots\cup\mathbf{B}(\lambda_m,t), \quad (20)$$

where λ_i is i-th path parameter:

$$\lambda_i = \begin{pmatrix} \mathbf{q}_i^T & \mathbf{h}_i^T & \mathbf{q}_{i+1}^T \end{pmatrix}^T, \quad i=0,\ldots,m. \quad (21)$$

Note that the two end states of the path must be fixed to the initial and goal states:

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

and an intermediate joint state $\mathbf{q}_i$, $i=1,\ldots,m$ is commonly used by $\mathbf{B}^{[i-1]}(t)$ and $\mathbf{B}^{[i]}(t)$. So, free variable vector to represent the global path is given by

$$\mathbf{X} = \begin{pmatrix} \mathbf{h}_0^T & \mathbf{q}_1^T & \mathbf{h}_1^T & \cdots & \mathbf{h}_{m-1}^T & \mathbf{q}_m^T & \mathbf{h}_m^T \end{pmatrix}^T. \quad (22)$$

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

$$\text{Minimize: } J(\mathbf{X}) = \sum_{i=0}^{m} L^{[i]}(\lambda_i), \quad (23)$$

$$\text{subject to: } \left(\kappa_j^{[i]}\right)^2 < \kappa_{max}^2, \quad (24)$$

$$\gamma_j^{[i]} > s_{j-1}^{[i]} \text{ and } \gamma_j^{[i]} > s_j^{[i]}, \quad (25)$$

where $L^{[i]}(\lambda_i)$ is the cost function of $\mathbf{B}^{[i]}$ and given by

$$L^{[i]}(\lambda_i) = \sum_{j=0}^{k_i-1}\left[w_s\cdot s_j^{[i]} + w_\kappa\cdot\left(\kappa_j^{[i]}\right)^2 - w_\gamma\cdot\gamma_j^{[i]}\right]. \quad (26)$$

The number of sample states k_i is determined in proportion to the arc length of $\mathbf{B}^{[i]}$. Each term of (26) can be normalized by replacing the weighing factors with $\bar{w}_s = w_s\|\mathbf{p}_{init}-\mathbf{p}_{goal}\|$, $\bar{w}_\kappa = w_\kappa\kappa_{max}^2$ and $\bar{w}_\gamma = w_\gamma\gamma_{max}$:

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

Again, we adapt *MMA* to solve the optimization problem (23)-(25). Differentiating (26) yields

$$\frac{dL^{[i]}(\lambda_i)}{d\lambda_i} = \sum_{j=0}^{k_i-1}\left[w_s\frac{\partial s_j^{[i]}}{\partial\lambda_i} + 2w_\kappa\kappa_j^{[i]}\frac{\partial\kappa_j^{[i]}}{\partial\lambda_i} - w_\gamma\frac{\partial\gamma_j^{[i]}}{\partial\lambda_i}\right], \quad (27)$$

where $\partial s/\partial\lambda$, $\partial\kappa/\partial\lambda$, and $\partial\gamma/\partial\lambda$ are given by (15), (16), and (19), respectively. Stacking up the gradients yields the gradient of performance index:

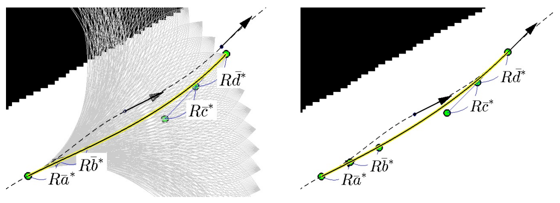
$$\frac{dJ(\mathbf{X})}{d\mathbf{X}} = \begin{pmatrix} \frac{\partial L^{[0]}}{\partial\mathbf{h}_0} \\ \frac{\partial L^{[0]}}{\partial\mathbf{q}_1} + \frac{\partial L^{[1]}}{\partial\mathbf{q}_1} \\ \vdots \end{pmatrix}. \quad (28)$$

Note that $\mathbf{h}_i$ only affects $\mathbf{L}^{[i]}$ while $\mathbf{q}_i$ affects both of $\mathbf{L}^{[i]}$ and $\mathbf{L}^{[i-1]}$ for it joins $\mathbf{B}^{[i]}$ and $\mathbf{B}^{[i-1]}$.

B. Merge

The solution searched through the state lattice (Fig. 1) plays a role as good initial guess to solve the *NLP* (23)-(25). However, not only accuracy of the initial guess but compactness of solution makes an high impact on establishing optimality or the rate of convergence. For the sake of the compactness, we merge consecutive Bézier curve primitives.

The first step is to compute offline the radius-normalized minimum curvature motion segment set. The set is obtained by solving the *NLP* (9)-(10) with $w_s=0$, for two end states such that the length between them is equal to one. Affine transformations such as translation, rotation, and scale can be applied on the curves by applying the respective transform on the control points of the curves.



(a) Lookup (b) Merge

Fig. 6. Merge process on two consecutive path segments.

Algorithm 1 takes as input the normalized motion segment set $\mathcal{K}$ with the path parameter consisting of joint states and control distance vectors, $\{\mathbf{q}_0,\mathbf{h}_0,\ldots,\mathbf{h}_m,\mathbf{q}_{m+1}\}$ and returns the reduced size of the parameter by merging adjacent Bézier curve primitives in the bottom-up style. The input path is recursively divided into two sub-paths and conquered (Line 7 and 8). Then the merged resulting sub-paths $left$ and $right$ are merged together by sub-routine *Merge* (Line 9). Denoting $\tilde{c}$ the index of the joint state between $left$ and $right$, the sub-routine looks up the closest control connecting $\mathbf{q}_{\tilde{c}-1}$ and $\mathbf{q}_{\tilde{c}+1}$ (Line 13). This is done by translating and rotating $\mathcal{K}$ to $\mathbf{q}_{\tilde{c}-1}$ and stretching by $R=\|\mathbf{p}_{\tilde{c}+1}-\mathbf{p}_{\tilde{c}-1}\|$ (Fig. 6(a)). Then the closest control $\mathbf{h}^* = R\bar{\mathbf{h}}^*$ can be interpolated for the resulting path segment to precisely reach at $\mathbf{q}_{\tilde{c}+1}$. However, we found that the Bézier curve $\mathbf{B}^*$ constructed by $\{\mathbf{q}_{\tilde{c}-1},\mathbf{h}^*,\mathbf{q}_{\tilde{c}+1}\}$

Algorithm 1 Merge Bézier Curves Comprising A Path

```
1: procedure MergeCurves($\{\mathbf{q}_0, \mathbf{h}_0, \dots, \mathbf{h}_m, \mathbf{q}_{m+1}\}$)
2:     $\mathcal{K}$ is normalized motion segment set.
3:     if $m \le 0$ then
4:         return $\{\mathbf{q}_0, \mathbf{h}_0, \dots, \mathbf{h}_m, \mathbf{q}_{m+1}\}$
5:     else
6:         $\tilde{c} \Leftarrow \lfloor \frac{m+2}{2} \rfloor$
7:         $left \Leftarrow MergeCurves(\{\mathbf{q}_0, \mathbf{h}_0, \dots, \mathbf{h}_{\tilde{c}-1}, \mathbf{q}_{\tilde{c}}\})$
8:         $right \Leftarrow MergeCurves(\{\mathbf{q}_{\tilde{c}}, \mathbf{h}_{\tilde{c}}, \dots, \mathbf{h}_m, \mathbf{q}_{m+1}\})$
9:         return $Merge(left, right)$
10:    end if
11: end procedure
12: procedure Merge($left = \{\mathbf{q}_{c_0}, \dots, \mathbf{h}_{\tilde{c}-1}, \mathbf{q}_{\tilde{c}}\}$, $right = \{\mathbf{q}_{\tilde{c}}, \mathbf{h}_{\tilde{c}}, \dots, \mathbf{q}_{c_f}\}$)
13:    $\mathbf{h}^* \Leftarrow LookUpClosestControl(\mathbf{q}_{\tilde{c}-1}, \mathbf{q}_{\tilde{c}+1}, \mathcal{C})$
14:    $\mathbf{B}^* \Leftarrow \mathbf{B}(\mathbf{q}_{\tilde{c}-1}, \mathbf{h}^*, \mathbf{q}_{\tilde{c}+1})$
15:    if $CollisionFree(\mathbf{B}^*) = true \wedge \kappa(\mathbf{B}^*) < \kappa_{max}$ then
16:        return $\{\mathbf{q}_{c_0}, \dots, \mathbf{q}_{\tilde{c}-1}, \mathbf{h}^*, \mathbf{q}_{\tilde{c}+1}, \dots, \mathbf{q}_{c_f}\}$
17:    else
18:        return $\{\mathbf{q}_{c_0}, \dots, \mathbf{h}_{\tilde{c}-1}, \mathbf{q}_{\tilde{c}}, \mathbf{h}_{\tilde{c}}, \dots, \mathbf{q}_{c_f}\}$
19:    end if
20: end procedure
```

is nearly same as the interpolated solution (Line 14 and Fig. 6(b)). If $\mathbf{B}^*$ is collision-free and subject to the robot's system limit (Line 15), then *left* and *right* are merged by combining the two with substituting $\mathbf{h}^*$ for $\{\mathbf{h}_{\tilde{c}-1}, \mathbf{q}_{\tilde{c}}, \mathbf{h}_{\tilde{c}}\}$ (Line 16). Otherwise, the two are simply joined at $\mathbf{q}_{\tilde{c}}$ (Line 18). Next section demonstrates the effectiveness of the *merge* process by showing the solution differences after optimization with and without merging (See Fig. 8).

V. Results

The proposed path planning algorithm was successfully implemented on our autonomous wheeled loader, the GIM mobile machine (Fig. 7). The robot is an articulated-frame-steering (AFS) machine joining front and rear bodies by a hydraulic cylinder. Steering is executed by elongation of the cylinder. (See [11] for more details of the kinematic model.)

In the field test, global scale geographic information, such as buildings, trees or vegetation area, were priori while obstacles such as traffic cones and trash bins were detected online. The robot initially plans the paths with the priori and replans over incrementally updated laser data. The workspace was encoded into the binary image of the obstacle map with the size of $150m \times 80m$ and the resolution of $0.2m$. The state lattice planner discretized the state space by $1m$ position resolution and 16 discrete headings. Paths found by the lattice planner undergo errors at the end states due to discretization. However, path optimization compensates the errors to precisely perform the desired task. Path computation was fast enough to be online replanning. Typical running times involving A^* search in the state lattice and optimization were less than $100ms$. Given the planned path, the steering control commands are computed by our prior work [11]. Fig. 7 shows snapshots of the robot operating in the field, which demonstrate realtime reactive navigation against dynamically changing obstacles.

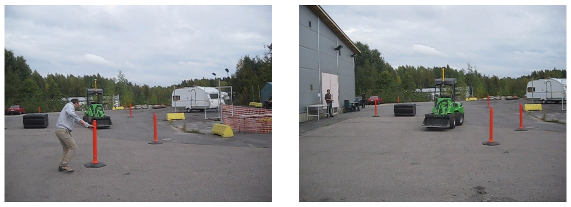


Fig. 7. Autonomous driving test. The robot reactively navigates against dynamically changing obstacles.

Let us demonstrate performance improvement by applying our planner compared to work of Dolgov et al. [9]. Main difference between the two methods is in parametric path representation. As in (22), our method uses joint states and control distance vectors to represent a piecewise Bézier curves path. On the other hand, Dolgov et al. use the discrete of coordinates to represent a piecewise linear path. From now on, we denote *BPO* (Bézier curves parametric Path Optimization) our method and *CPO* (Coordinates parametric Path Optimization) the method of [9].

CPO performs two-stage optimization procedure. In the first stage, *NLP* is formulated as (23)-(25) on the coordinates of the vertices to parameterize a path. Then, the method interpolates the first-stage solution by performing another optimization with higher resolution path discretization while holding the original vertices fixed.

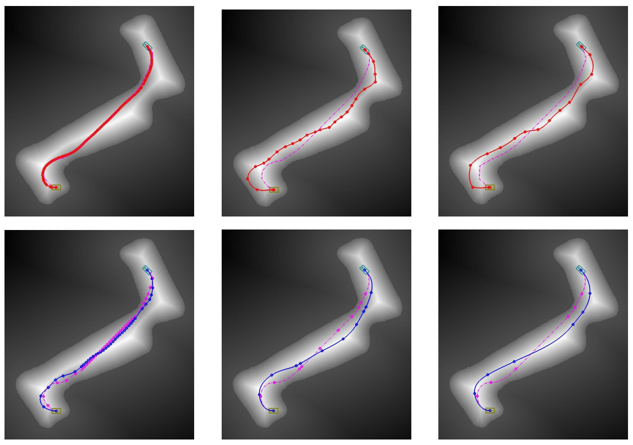


Fig. 8. Given an identical initial guess (dashed magenta curves), optimal solutions by *CPO* (red solid curves in top row) and *BPO* (blue solid curves in bottom row) are provided. The sizes of $\mathbf{X}$ in each column are same.

Fig. 8 visualizes the effectiveness of *BPO* compared to *CPO*. Given an identical initial guess (dashed magenta curve), the figure compares solutions of the *NLP* found by *CPO* (red solid curves in top row) and *BPO* (blue solid curves in bottom row), depending on size of the path parameter $\mathbf{X}$. The initial guess is a sequence of Bézier curves found in the *state lattice*. In the case of *BPO*, the size of $\mathbf{X}$ decreases by processing *merge* with the depth of 0, 3, and 6 from left to right. The depth of d specifies maximum number of recursion such that 2^d adjacent primitives are merged at most. Each column of *CPO* sampled equally spaced vertices on the path such that the size of **X** is equal to that of *BPO*.

Both methods are common in that the optimized curves deform more as the size of **X** decreases. *CPO* reveals its drawback that generates sharp corners as the path vertices become too coarse. These can be smoothed by performing multiple-stage interpolations, but it will blow up computation. On the contrary, *BPO* paths converge to better solutions while maintaining smoothness on each primitive curve as the size of **X** decreases by *merge*. In addition, the reduced size of **X** leads to faster computation. In this example, the solution with merging (bottom-right) was computed approximately 10 times faster than the solution without merging (bottom-left).

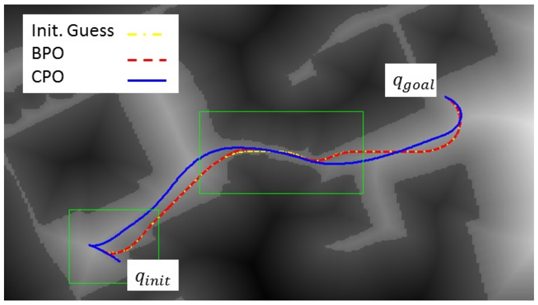


Fig. 9. Comparison of global path optimization by *CPO* (red dashed) and *BPO* (blue solid) for a given initial guess (yellow dashed dot).

Fig. 9 more clearly shows the superiority of *BPO* in a larger and more complex workspace. While the *CPO* solution (red dashed curve) is confined to sub-optimum near the initial guess (yellow dashed dot), the *BPO* solution (blue solid curve) converges to better local optimum. The boxed bottom-left area highlights another benefit of *BPO* in cusp-point optimization. The cusp-point indicates the point where the robot switches its moving directions. *CPO* splits up a path into forward/backward segments, then anchors the points down as boundary conditions and optimizes each one independently. This approach can cause computation to blow up as number of the points increases. On the other hand, *BPO* adjusts the point and all the segments can be optimized by once. The resulting solution is not only calculated faster, but achieves better optimality. In addition, the boxed middle area clearly shows *BPO* generates closer-to-the-voronoi-diagram thereby safer solution than *CPO* at a narrow passage.

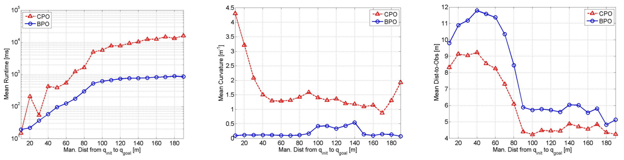


Fig. 10. Performance comparison between *CPO* (red) and *BPO* (blue) in terms of runtime (left), magnitudes of curvatures (middle), and proximity-to-obstacle (right).

Fig. 10 provides the numerical comparison of planning performances for 4000 random generated test cases in an identical work environment. Both methods have been implemented in same programming infrastructure and *NLP* parameters. The initial guesses are paths searched in the state lattice. *CPO* samples equally spacing vertices on the paths. As shown in Fig. 8, the sampling rates to match the size of **X** to *BPO* may generate sharp corners and violate the maximum curvature constraint. In this simulation, it leads to 56% violation rate. So, we used the sampling interval of $1m$ and $.2m$ for the first and second stage optimization, as suggested in [9]. *BPO* (blue circles) outperforms *CPO* (red triangles) in terms of all the factors of runtime (left), magnitudes of curvatures (middle), and distances-to-nearest-obstacle (right). While *CPO* takes $3.28s$ to compute a path in average, *BPO* only takes $300ms$. The improvement in magnitudes of curvatures and distances-to-nearest-obstacle are greater than a factor of 8.9 and 1.295, respectively.

VI. Conclusions

This paper proposes a new efficient path planning algorithm. The benefits of the algorithm are summarized as:

- Feasible controls are spanned with few parameters.
- The gradient of the cost involving arc-length, smoothness, and collision clearance is efficiently computed.
- The differential constraint can be satisfied without requiring any interpolation process.
- The switching-direction-points have degrees of freedom and can be processed in one stage optimization.

The experimental results demonstrate not only successful generation of safe routes but comfortable control results for autonomous ground vehicles in realtime.

References

[1] K. Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM Journal on Optimization*, pages 555–573, 2002.

[2] S. Koenig and M. Likhachev. D* lite. In *the AAAI Conference of Artificial Intelligence (AAAI)*, pages 476–483, 2002.

[3] D. Ferguson and A. Stentz. Using interpolation to improve path planning: The field d^* algorithm. *Journal of Field Robotics*, 23(2):79–101, February 2006.

[4] B. Lau, C. Sprunk, and W. Burgard. Kinodynamic motion planning for mobile robots using splines. In *the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2427–2433, 2009.

[5] J.-W. Choi, R. E. Curry, and G. H. Elkaim. Continuous curvature path generation based on bézier curves for autonomous vehicles. *IAENG International Journal of Applied Mathematics*, 40(2), 2010.

[6] M. Pivtoraiko and A. Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3231–3237, August 2005.

[7] M. Likhachev and D. Ferguson. Planning long dynamically-feasible maneuvers for autonomous vehicles. *International Journal of Robotics Research (IJRR)*, 28:933–945, 2009.

[8] A. Kelly and B. Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *The International Journal of Robotics Research*, 22(7-8):583–601, July 2003.

[9] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010.

[10] N. Ratliff, M. Zucker, J.A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation*, pages 489–494, May 2009.

[11] R. Ghabcheloo, M. Hyvönen, J. Uusisalo, O. Karhu, J. Järä, and K. Huhtala. Autonomous motion control of a wheel loader. In *ASME 2009 Dynamic Systems and Control Conference*, 2009.