JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS
Vol. 36, No. 6, November–December 2013

On-Line Path Generation for Unmanned Aerial Vehicles Using B-Spline Path Templates

Dongwon Jung*
Korea Aerospace Research Institute, Daejeon 305-333, Republic of Korea
and
Panagiotis Tsiotras†
Georgia Institute of Technology, Atlanta, Georgia 30332-0150

DOI: 10.2514/1.60780

The problem of generating a smooth reference path, given a finite family of discrete, locally optimal paths, is investigated. A finite discretization of the environment results in a sequence of obstacle-free square cells. The generated path must lie inside the channel generated by these obstacle-free cells, while minimizing certain performance criteria. Two constrained optimization problems are formulated and solved subject to the given geometric (linear) constraints and boundary conditions in order to generate a library of B-spline path templates offline. These templates are recalled during implementation and are merged together on the fly in order to construct a smooth and feasible reference path to be followed by a closed-loop tracking controller. Combined with a discrete path planner, the proposed algorithm provides a complete solution to the obstacle-free path-generation problem for an unmanned aerial vehicle in a computationally efficient manner, which is suitable for real-time implementation.

Nomenclature

$\boldsymbol{b}(u)$	=	two-dimensional B-spline curve parameterized by the knot u
$\boldsymbol{b}_j$	=	jth control points of two-dimensional B-spline curve
$c_{i,j}$	=	cell at (i, j) location
e_L, e_R	=	left and right bounding envelopes of a two-dimensional B-spline curve
h	=	cell size
H^k	=	convex hull of S^k and S^{k+1}
$\mathcal{J}(\cdot,\cdot)$	=	cost function
$\mathcal{L}(\cdot,\cdot)$	=	linear interpolation operator
ℓ_L^k, ℓ_R^k	=	kth line segments of the left and right bounding envelopes
n_{t_L}, n_{t_R}	=	number of concave corner points of t_L and t_R
$N_j^d(u)$	=	B-spline basis function of degree d
$\boldsymbol{P}(u), \boldsymbol{Q}(v)$	=	two-dimensional B-spline curves to be merged
$\boldsymbol{P}_i, \boldsymbol{Q}_i$	=	control points of $\boldsymbol{P}(u)$ and $\boldsymbol{Q}(v)$
$\boldsymbol{R}(w)$	=	merged two-dimensional B-spline curve parameterized by the knot w
S^k	=	axis-aligned bounding box at the kth Greville abscissa
t_L, t_R	=	left and right channel polygons
$\{u_k\}$	=	nondecreasing knot sequence
u_L^k, u_R^k	=	feature points obtained by intersecting two consecutive convex hulls
u^*	=	Greville abscissa
v_i^k	=	line segments connecting the corners of S^k and $S^{k+1} (i = 1, \ldots, 4)$
w_m	=	merging knot
α	=	weight constant
δ_i	=	perturbation from the control points of $\boldsymbol{Q}_i$
ϵ_i	=	perturbation from the control points of $\boldsymbol{P}_i$
κ	=	curvature of the curve
ψ	=	tangent direction

Presented as Paper 2008-7135 at the AIAA Guidance, Navigation and Control Conference and Exhibit, Honolulu, Hawaii, 18–21 August 2008; received 16 October 2012; revision received 14 February 2013; accepted for publication 14 February 2013; published online 22 July 2013. Copyright © 2013 by Dongwon Jung and Panagiotis Tsiotras. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission. Copies of this paper may be made for personal or internal use, on condition that the copier pay the $10.00 per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923; include the code 1533-3884/13 and $10.00 in correspondence with the CCC.

*Senior Researcher, Mission Design Department; djung@kari.re.kr. Member AIAA.

†Dean's Professor, School of Aerospace Engineering; tsiotras@gatech.edu. Fellow AIAA (Corresponding Author).

I. Introduction

THE autonomous guidance and navigation of mobile vehicles (e.g., "agents") is an active research topic owing to its importance for the development of intelligent, fully autonomous vehicles with increased decision-making capabilities. For the case of unmanned aerial vehicles (UAVs) in particular, the ability of fully automated guidance and navigation allows UAVs to accomplish missions under various scenarios and with minimal human intervention. Because of the stringent operational requirements and the restrictions imposed on UAVs for autonomy, safety, and efficiency, fully autonomous guidance and navigation (including path planning, path generation, and path following) for low-cost UAVs with limited onboard computational and power resources is a real challenge.

The guidance and navigation of aerial vehicles is being traditionally accomplished by using a hierarchical control structure [1–3], which may include path-planning, path-generation, and path-following control layers. Typically, a simple tracking controller operates over a sequence of waypoints, which are computed by a (usually human) mission planner. Because the waypoint generation problem is done at a geometric level (e.g., avoid obstacles in the environment, visit certain areas of interest) there is no a priori guarantee that an arbitrary path connecting these waypoints will be flyable, especially for fixed-wing UAVs. In other words, the geometric and trajectory generation layers of the problem are decoupled in a typical hierarchical control structure. The lower-level trajectory tracking controller is given the task to fly the vehicle while trying to follow the given path. There is no a priori guarantee that the path to be followed is feasible, however. This dichotomy resulting from neglecting the vehicle dynamics during the path-generation step leads to suboptimal performance. Optimal (and feasible) trajectories can be generated using a plethora of available trajectory generation algorithms, but these solutions are typically computationally expensive, have convergence issues, and are therefore not amenable to on-line, embedded execution.

A better solution to the guidance and navigation control problem should take into account the dynamic constraints of the vehicle, resulting in a smooth, flyable path passing through the given waypoints. Furthermore, the waypoints must be connected such that the generated smooth path preserves the continuity of the curvature between successive arc segments, while minimizing the maximum curvature [4,5]. In [6], the authors proposed a dynamic trajectory smoothing algorithm by which the path segments are smoothed to yield an extremal trajectory with explicit consideration of the kinematic constraints of a fixed-wing UAV. The preceding references deal with smoothness with respect to the continuity of curvature and curvature derivative. However, careful consideration should be taken to preserve the continuity among line and arc segments.

To avoid such problems, in this paper, piecewise polynomial B-splines are used as path templates to generate a feasible path. By incorporating a high-level path-planning algorithm such as $\mathcal{D}^*$-lite [7], one obtains a numerically efficient on-line path-smoothing algorithm by merging together a set of B-spline path templates in order to generate a smooth reference path.

Splines have been widely adopted when computing smooth, dynamically feasible trajectories for UAVs. A series of cubic splines was employed in [8] to connect straight line segments in a near-optimal manner. The authors of [9] presented an implicit time parameterization of the trajectory using a B-spline representation. The design of an obstacle-avoiding B-spline path has been dealt with by Berglund et al. in [10], whereas the real-time modification of a spline path has been proposed in [11]. Most recently, the authors of [12] proposed an algorithm for constructing collision-free paths using cubic B-splines having bounded, continuous curvatures. Bézier splines were also used in [13,14] to generate trajectories that satisfy dynamic constraints of autonomous vehicles. The advantage of employing (B-)splines for generating a smooth path stems from the fact that the path can be represented using a small number of parameters. This offers significant benefits when encoding a smooth path using splines and also when optimizing the path shape. Splines are therefore ideally suitable for onboard implementation.

The path-generation problem using B-splines involves finding the solution of a constrained optimization problem not only to avoid forbidden regions but also to generate flyable trajectories. In [15], a polygonal channel composed of piecewise polygonal lines (polylines) is assumed to be given. A B-spline curve that remains within the channel is found by quadratic programming. This was made possible by adopting tight linear envelopes for the splines [16], by which a B-spline is represented as an approximate bounding polygon. In this paper, the results of [15,16] are extended in two dimensions by incorporating two-dimensional (2-D) B-spline curves in the associated constrained optimization problem. Instead of smoothing the entire path from the initial position to the final goal, one smooths the path segments over a finite planning horizon with respect to the current position of the UAV. To this end, one first generates a set of all possible combinations of discrete path sequences within a finite planning horizon. One thus obtains the corresponding channel constraints; the path templates are computed by (offline) optimization. These path templates are stored in a library for on-line implementation. Each B-spline segment from these path templates is merged to the existing B-spline path to generate a reference path on-line, while preserving second-order continuity along the entire path.

The proposed approach has the benefit of generating a reference path that is appropriate for a complex environment with obstacles. When compared with previous approaches, for instance [6,8,11], the proposed approach has the following advantages. First, it can easily handle obstacles in the environment, whereas the extremal trajectories between the waypoints in [6] were computed without consideration to obstacles in the environment. In addition, our algorithm is fast and generates a feasible path, whereas the smoothing algorithm proposed in [8] requires on-line optimization, which is typically a computationally expensive operation. The proposed algorithm has similarities with the algorithm proposed in [11], in which B-splines are also used to generate smooth paths. However, contrary to our approach, in [11], the path was defined using a single B-spline. Real-time modification therefore implies the recalculation of the relevant control points in order to avoid obstacles. If the length of the path becomes too long, then the modification of a B-spline path might become cumbersome. In our case, the on-line modification of the reference path for a dynamically changing environment is dealt with by incrementally merging the template B-splines in accordance to a high-level path planner, which is a fast and numerically reliable process.

II. Tight Envelopes for B-Spline Curves

In this section, a brief description of the tight envelope generation for one-dimensional (1-D) B-spline curves is given along with an extension of the results to two dimensions. This envelope provides quantitative bounds of all allowable B-spline curves with respect to the given control points. In [16], the authors derived explicit bounds for the 1-D case, proving that the bounds are tight and quantitative enough to provide the B-spline's salient information [17]. The same authors also showed that these bounds are piecewise linear. These envelopes are crucial because they will be used in Sec. III to simplify the optimization problem of finding the best (e.g., minimum length, minimum curvature) spline inside an obstacle-free channel.

A. Tight Envelopes for B-Spline Functions

In this section, the construction of tight envelopes for 1-D B-splines [16] is briefly reviewed. This will allow the extension of the envelope construction to two dimensions, which is given in the next section. A 1-D B-spline $b(u)$ is expressed by [18]

$$b(u) = \sum_{j=0}^{m} b_j N_j^d(u) \tag{1}$$

where b_j are the control points and $N_j^d(u)$ are the B-spline basis functions of degree d, which are defined over a nondecreasing knot sequence $\{u_k\}_{k=0}^{m+d+1}$ such that $u_0 \le u_1 \le \cdots \le u_{m+d+1}$. The number of knots is determined by the sum of the number of control points $(m+1)$ and the B-spline order $(d+1)$. The first and the last knots of the sequence should have multiplicity $(d+1)$ for a B-spline to pass through the first and the last control points; that is, $u_0 = u_1 = \cdots = u_d$ and $u_{m+1} = u_{m+2} = \cdots = u_{m+d+1}$, respectively. The B-spline basis functions are computed by the well-known Cox–de Boor recursion formulas [18].

Among the several useful properties of the B-spline basis functions is their local support property [18], which offers flexibility in terms of curve design because it allows local modification of a B-spline curve without changing the entire shape of the curve.

To obtain bounds on the allowable splines, let the control polygon of the B-spline $\ell(u)$ be defined by piecewise line segments connecting the control points b_j at the Greville abscissae [19]:

$$u_j^* = \sum_{i=j+1}^{j+d} u_i/d \tag{2}$$

such that at each Greville abscissa the equation $\ell(u_j^*) = b_j$ is satisfied. Accordingly, the envelope of the B-spline specifies a bound on the distance between $b(u)$ and its control polygon $\ell(u)$. Hence, this envelope provides a good estimate of the shape of the B-spline by carrying most of the salient information about the curve itself. The envelopes derived in [16] are expressed in terms of the weighted second difference of the control points,

$$\Delta_2 b_j \triangleq b'_{j+1} - b'_j, \qquad b'_j = \frac{b_j - b_{j-1}}{u_j^* - u_{j-1}^*} \tag{3}$$

and by the nonnegative convex functions over the interval $[u_k^*, u_{k+1}^*]$ $(k = 0, 1, \ldots, m)$ given as follows:

$$\beta_{ki}(u) = \begin{cases} \sum_{j=1}^{\bar{k}} (u_j^* - u_i^*) N_j^d(u) & i > k, \\ \sum_{j=\underline{k}}^{i} (u_i^* - u_j^*) N_j^d(u) & j \le k \end{cases} \tag{4}$$

where $\underline{k}$ and $\bar{k}$ denote, respectively, the index of the first and the last B-spline basis functions that are nonzero over the corresponding interval. The distance between the spline function $b(u)$ and its control polygon $\ell(u)$ is then calculated as follows [20]:

$$b(u) - \ell(u) = \sum_{i=\underline{k}}^{\bar{k}} \Delta_2 b_i \beta_{ki}(u) \tag{5}$$

Because the β_{ki} are nonnegative and convex functions over the corresponding interval $[u_k^*, u_{k+1}^*]$, their maxima occur at each endpoint of the interval, i.e., at each Greville abscissa. Then, the piecewise linear functions $\underline{e}(u)$ and $\bar{e}(u)$ that interpolate their values at each Greville abscissa can be employed to represent tight envelopes of the spline function [20]. Subsequently, the maximal bounds from the B-spline function to its control polygon are obtained by the simple inequalities

$$\underline{e}(u) \le b(u) \le \bar{e}(u) \tag{6}$$

Figure 1 depicts a cubic B-spline function $b(u)$ over the knot sequence $u \in [0, 1]$. The control polygon is drawn by a dashed line, whereas the bounding envelopes $\underline{e}(u)$ and $\bar{e}(u)$ are drawn by dotted and dashed-dot lines, respectively.

B. Tight Envelopes for Planar B-Spline Curves

The extension to the 2-D case follows easily from the results in the preceding section. A 2-D planar B-spline curve $\boldsymbol{b}(u) = [b^x(u) b^y(u)]^\top$ is expressed in terms of the B-spline basis functions as follows:

$$\boldsymbol{b}(u) = \sum_{j=0}^{m} \boldsymbol{b}_j N_j^d(u) \tag{7}$$

where $\boldsymbol{b}_j = [b_j^x b_j^y]^\top$ are the corresponding control points in the x and y directions. It will be assumed that the B-spline curve is clamped at the first and last control points by assigning the $(d+1)$ multiple knots at the first and last knots.

At each Greville abscissa, u_k^*, the 1-D bounds in Eq. (6) generate a 2-D bounding box for which the x and y axes are determined by the 1-D envelope, as follows:

$$\underline{e}^i(u_k^*) \le b^i(u_k^*) \le \bar{e}^i(u_k^*), \qquad i = x, y \tag{8}$$

Let the axis-aligned bounding box at the kth Greville abscissa be denoted by S^k. Then, the curve segment $\boldsymbol{b}(u)$, for $u \in [u_k^*, u_{k+1}^*]$, lies in the convex combination of S^k and the consecutive box S^{k+1}, owing to the linearity of $\underline{e}(u)$ and $\bar{e}(u)$. This is denoted by a linear interpolation between two bounding boxes:

$$H^k = \mathcal{L}(S^k, S^{k+1}) \tag{9}$$

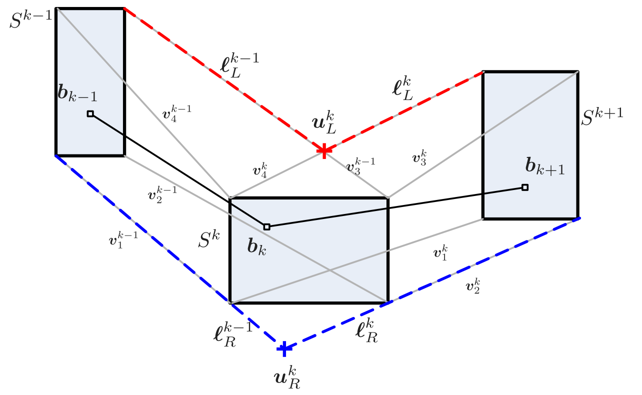


Fig. 2 Constructing the bounding envelopes of a planar curve from neighboring bounding boxes. The bounding envelope is characterized by a union of convex hulls, which is represented by piecewise linear polygonal lines.

Note that H^k is the convex hull of S^k and S^{k+1}, which is circumscribed by the edges of S^k and S^{k+1} and the lines that connect the corners of S^k and S^{k+1}. Let $\boldsymbol{v}_i^k$ $(i = 1, \ldots, 4)$ be the line segments connecting the corresponding corners of S^k and S^{k+1}. Then, $\boldsymbol{v}_1^k$ connects the lower-left corner of S^k to the lower-left corner of S^{k+1}, $\boldsymbol{v}_2^k$ connects the lower-right corner of S^k to the lower-right corner of S^{k+1}, and so on. Figure 2 shows these line segments. As mentioned in the preceding sentences, the convex hull H^k over the knot $u \in [u_k^*, u_{k+1}^*]$ consists of parts of the edges of S^k and S^{k+1} and exactly two extra line segments chosen among $\boldsymbol{v}_i^k$. By the union of each convex hull H^k, where $k = 0, \ldots, m$, the 2-D bounding envelope of the B-spline curve is obtained; it is represented by two piecewise linear polygonal lines e_L and e_R, denoted by the left envelope and the right envelope with respect to the control polygon, respectively. These polylines are composed of a set of line segments ℓ_L^k and ℓ_R^k, where $k = 0, \ldots, m$, which are referenced by the feature points u_L^k and u_R^k, respectively. These feature points are obtained by intersecting two consecutive convex hulls; that is, the features points are chosen to be the intersecting points among $\boldsymbol{v}_j^k$.

It is possible, however, that two line segments do not intersect. Consider, for instance, the line segments $\boldsymbol{v}_1^{k-1}$ and $\boldsymbol{v}_2^k$ in Fig. 2. The feature point u_R^k is inferred from the extended line segments of $\boldsymbol{v}_1^{k-1}$ and $\boldsymbol{v}_2^k$; hence, the line segments ℓ_R^{k-1} and ℓ_R^k are obtained with respect to u_R^k. Consequently, the line segments ℓ_L^k and ℓ_R^k are joined together to form piecewise linear envelopes of the B-spline curve e_L and e_R, respectively. Figure 3 shows an example of the 2-D bounding envelopes of a given B-spline curve, which reveals that the entire B-spline curve stays inside the envelope generated by e_L and e_R.

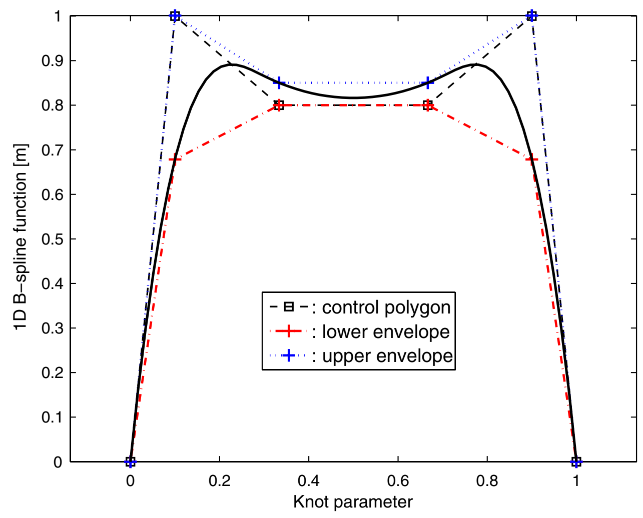


Fig. 1 1-D cubic B-spline bounding envelopes. The control polygon stays inside the bounding envelopes, and so does the B-spline curve.

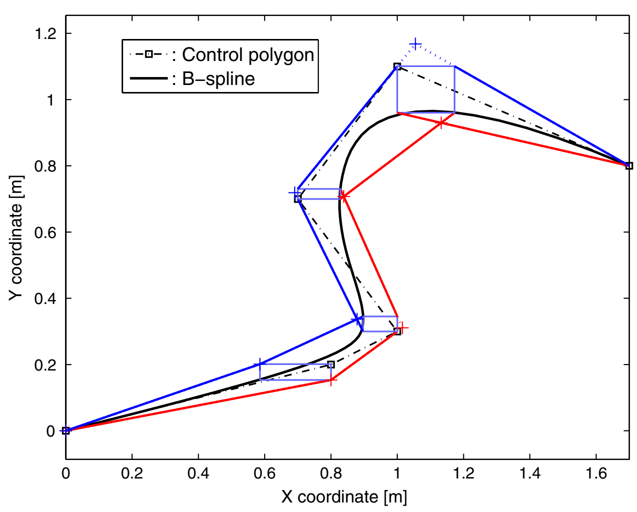


Fig. 3 Bounding envelopes e_L and e_R of 2-D cubic B-spline.

III. Obstacle-Avoidance Path Optimization

In this section, B-spline curves are employed as primitives for path design. A constrained optimization problem is formulated by constructing channel constraints, ensuring that the designed path stays inside the given polygonal channel, which is assumed to be obstacle free. Section IV, later on, describes how such a channel is generated by connecting sequences of obstacle-free cells in the environment using a graph search algorithm, such as $\mathcal{A}^*$, $\mathcal{D}^*$, or $\mathcal{D}^*$-lite.

A. Channel Constraints for Obstacle Avoidance

In the sequel, the term channel refers to a feasible region in the environment over which a geometric path will be optimized. It is assumed that two nonintersecting polylines constitute a channel, separating the feasible region from the obstacle region. Hence, one can compute a path that avoids obstacles, while satisfying the given performance criteria.

For the 1-D B-spline function optimization case [15], and given nonintersecting channel polygons, the inequality constraints that ensure feasibility in terms of the lower and upper envelopes $\underline{e}(u)$ and $\bar{e}(u)$ are given in Eq. (6). These inequalities provide bounds with respect to the chosen parameters of the B-spline function (namely, the control points) such that the B-spline function stays between the given polygons. For the 2-D case, because one is dealing with a B-spline curve and not a B-spline function, the channel constraints should be formulated in terms of geometric constraints, as opposed to the linear inequalities with respect to the control point parameters. Nonetheless, it is crucial that these constraint equations capture the condition that the given geometric channel polygon contains the entire envelope of the B-spline curve.

To this end, let $\mathcal{O} \subset \mathbb{R}^2$ denote the obstacle region. A signed distance-map function $f(\boldsymbol{x}; \ell)$, $\boldsymbol{x} \in \mathbb{R}^2$ with respect to a polygonal line ℓ is introduced in order to provide a relative distance metric for formulating geometric constraints as follows:

$$f(\boldsymbol{x}; \ell) \triangleq s \min\{d_1, d_2\} \tag{10}$$

where $d_1 \in \mathcal{D}_1 = \{\|\boldsymbol{x} - \boldsymbol{c}_i\|, i = 0, \cdots, q\}$ is the distance from a point x to a corner point c_i of the polygonal line ℓ, $d_2 \in \mathcal{D}_2 = \{d(\boldsymbol{x}, \ell_i), i = 0, \cdots, q-1\}$ is the perpendicular distance from x to the line segment ℓ_i that connects two consecutive corner points c_i and c_{i+1}, and s is a sign value that dictates the location of the point x with respect to ℓ as follows:

$$s = \begin{cases} +1, & \boldsymbol{x} \in \mathcal{O}, \\ 0, & \boldsymbol{x} \in \ell, \\ -1, & \boldsymbol{x} \notin \mathcal{O} \end{cases} \tag{11}$$

Figure 4 shows the distance-map function with respect to an arbitrary zigzag-shape polygonal line. Far-away points from the line have

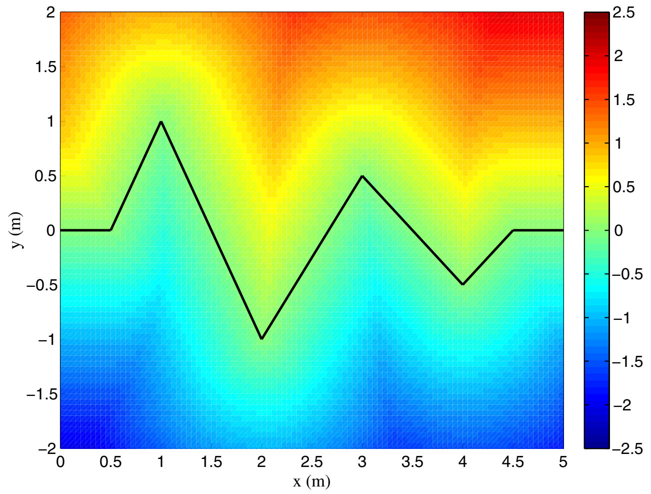


Fig. 4 Signed distance map for an arbitrary polygonal line. The feasible region is characterized by the negative function values.

bigger absolute values, whereas points close to the line yield smaller values close to zero. The information about the relative location of a point with respect to the polygonal line is determined by its sign.

To formulate inequality constraints similar to those in [15], one makes use of the fact that the envelopes of a planar B-spline are characterized by the feature points u_L^k and u_R^k of the piecewise linear envelopes e_L and e_R, shown in Fig. 2. It then follows that, if all the feature points are placed inside the feasible region, then each bounding box at $u = u_k^*$ (and, hence, also the envelopes of the B-spline curve) is most likely to be contained inside the feasible region as well. To this end, let t_L and t_R be the polygonal lines representing the obstacle boundaries. Using Eq. (10), the feature points u_L^k and u_R^k at each Greville abscissa $u = u_k^*$ should satisfy the following inequality expressions:

$$f(u_L^k; \boldsymbol{t}_L) \leq 0, \qquad f(u_R^k; \boldsymbol{t}_R) \leq 0 \tag{12}$$

where $k = 0, \ldots, m$.

In addition to inequalities [Eq. (12)], extra constraints are required to impose the conditions of complete inclusion of the envelopes inside the feasible channel. Recall that the envelope of the B-spline curve is derived by the convex hull of each bounding box, which results in the envelope being represented by piecewise polygonal lines. As a result, not only each bounding box at $u = u_k^*$ of the B-spline curve should be contained in the feasible region, as discussed in the preceding paragraph, but also the convex hull is required to be contained in the feasible region as well. Because the obstacle boundaries are also represented by polylines, in order to ensure that the convex hull of the B-spline envelope completely lies inside the channel, one must make sure that the corner points of the obstacle boundaries are placed outside the envelope. Accordingly, the obstacle region is excluded from the B-spline envelope. It should be noted that the convex corner points of the obstacle boundary are implicitly excluded from the envelope of the B-spline. Hence, one only applies this condition to the concave corners of t_L and t_R with respect to the obstacle region, as marked by the triangles in Fig. 5. This results in a set of inequality constraints using Eq. (10) at each concave corner point, in conjunction with the piecewise linear envelopes as follows:

$$f(\boldsymbol{c}_l^{t_L}; \boldsymbol{e}_L) \geq 0, \qquad f(\boldsymbol{c}_m^{t_R}; \boldsymbol{e}_R) \geq 0 \tag{13}$$

where $l = 1, \ldots, n_{t_L}$, where n_{t_L} is the number of concave corner points of t_L and $m = 1, \ldots, n_{t_R}$, and where n_{t_R} is the number of concave corner points of t_R. Note that the positive sign in Eqs. (13) implies that the corner points are excluded from the bounding envelope of the B-spline curve. Consequently, the inequality constraints in Eqs. (12) and (13) ensure that the envelope of the B-spline stays inside the channel, as depicted in Fig. 5.

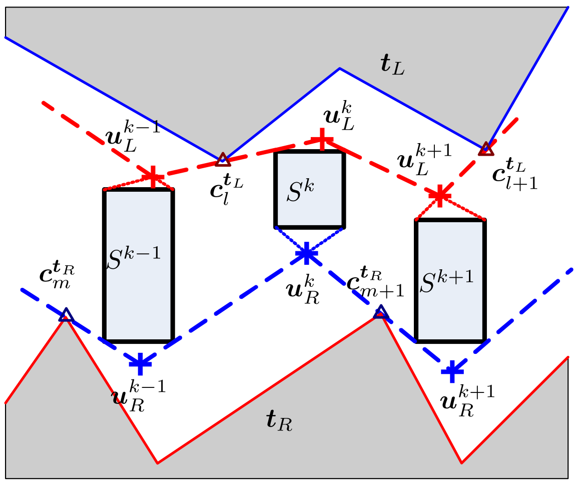


Fig. 5 Geometric constraints formulation. The channel is given by two polylines t_L and t_R, the envelope of the B-spline is drawn by the dashed lines, which is supposed to stay inside the channel.

B. Smooth Curve Optimization

In this section, the problem of designing a smooth curve using a quartic B-spline in terms of the knot parameter u is considered. A quartic B-spline preserves continuity up to the third-order derivative, thus resulting in the continuity of the derivative of the curvature. Without loss of generality, the knot parameter is selected as $u \in [0, 1]$, and the first and last knots have a multiplicity of five such that $u_0 = \cdots = u_4 = 0$ and $u_{m+1} = \cdots = u_{m+5} = 1$. Consequently, the B-spline curve will be clamped at, or pass through, the first and last control points.

One manipulates the $(m+1)$ control points of a B-spline curve given by $\boldsymbol{b}_j = [b_j^x b_j^y]^\top$ $(j = 0, \ldots, m)$. These have a direct influence on the shape of the curve. Because the shape of the curve is closely related to the given channel geometry that encloses the curve, and the optimization is affected by the number of control points, it is advisable to opt for the minimum number of control points, while taking into account the complexity of the given channel geometry. The knot sequence is chosen arbitrarily for some nondecreasing sequence of numbers in the interval (0, 1), so that the number of knots matches that of the control points. It should be noted that the number of knots and control points may be altered during the optimization process by inserting knots or control points if the envelopes of the B-spline curve need to be refined [15].

Two different performance indices are adopted in this work to optimize B-spline curves in terms of attaining two distinct objectives: in order to keep the B-spline curve as close as possible to a straight line (say, for instance, one wishes the UAV to travel with the maximum allowable speed), for which $\Delta_2 \boldsymbol{b}_j = 0$, one employs the cost function

$$\mathcal{J}_1(\{\boldsymbol{b}_j\}_{j=0}^m, \{u_k\}_{k=0}^{m+5}) = \sum_{j=1}^{m-1} (\Delta_2 \boldsymbol{b}_j)^\top (\Delta_2 \boldsymbol{b}_j) \tag{14}$$

The performance index [Eq. (14)] minimizes the curvature variation of the resulting B-spline. On the other hand, note that the arc length of the B-spline curve is approximately captured by the total length of the control polygon $\ell(u)$. Hence, for a shortest path, one employs a cost function that is the length of the piecewise control polygon as follows:

$$\mathcal{J}_2(\{\boldsymbol{b}_j\}_{j=0}^m) = \sum_{j=0}^{m-1} \|\ell_j\|_2 \tag{15}$$

The constraints for the optimization problem consist of both equality and inequality constraints. The equality constraints stipulate boundary conditions for the position, tangent direction, and curvature at each endpoint at $u_0 = 0$ and $u_{m+5} = 1$ as follows:

$$\boldsymbol{b}(0) = \boldsymbol{p}_0, \qquad \boldsymbol{b}(1) = \boldsymbol{p}_f \tag{16a}$$

$$\psi(0) = \psi_0, \qquad \psi(1) = \psi_f \tag{16b}$$

$$\kappa(0) = \kappa_0, \qquad \kappa(1) = \kappa_f \tag{16c}$$

where $\psi(u)$ and $\kappa(u)$ are the tangent direction and the curvature of the B-spline curve at each knot u, respectively. The channel constraints in Eqs. (12) and (13) become the inequality constraints for the optimization problem.

Using the ingredients discussed so far, the path optimization problem can then be stated as follows. Given a knot sequence $\{u_k\}_{k=0}^{m+5}$, two polygonal lines for the channel geometry, and boundary conditions for each endpoint, find a B-spline curve, which minimizes the cost function in Eq. (14) or (15), subject to the equality constraints in Eqs. (16) and the inequality constraints in Eqs. (12) and (13).

A standard sequential quadratic programming solver for this optimization problem has been used [21]. Figure 6a shows the optimization result using the cost function in Eq. (14). The constructed quartic B-spline curve is drawn by a solid line, and the bounding envelopes are drawn by dashed and dashed-dot lines. The B-spline curve, as well as the envelopes, stay inside the specified channel polygon. For the case of the shortest curve that involves the cost function in Eq. (15), Fig. 6b reveals that the computed B-spline curve is indeed shorter than the preceding case, albeit the maximum curvature $|\kappa|_{\max}$ is a bit larger than that of the preceding case (equivalently, one might want to compare the curvature "energy" $\int \kappa^2 \, \mathrm{d}s$ along the curve). In the following, a combined cost function has been used, emphasizing the shortest path criterion in Eq. (15).

IV. Path Templates for Obstacle-Free Channels

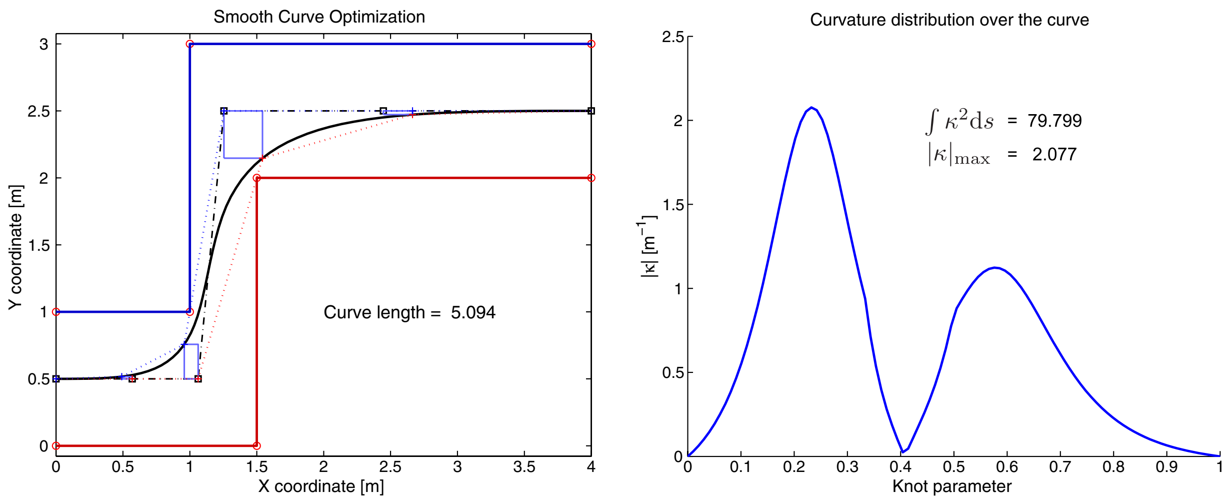
In this section, a library of "path templates" is constructed, which are to be used as the elementary path primitives for on-line path generation. These templates contain a set of planar B-spline curves, which will be regarded as local path segments to smooth a discrete path sequence. It is assumed that the obstacle-free discrete path sequence is provided by a high-level path planner, such as the multiresolution path-planning algorithm proposed in [22,23], or a similar discrete search algorithm, such as $\mathcal{A}^*$ or $\mathcal{D}^*$. The algorithm constructs an obstacle-free channel such that the discrete path sequence is represented by a series of square cells. For all possible channels that correspond to path sequences over a finite planning horizon, a set of B-spline curves included in this channel is computed via the path optimization, as discussed in the preceding section. Different channel constraints and boundary conditions are imposed in each case, resulting in a different set of smooth local path primitives.

A. Path Rules Within a Finite Horizon

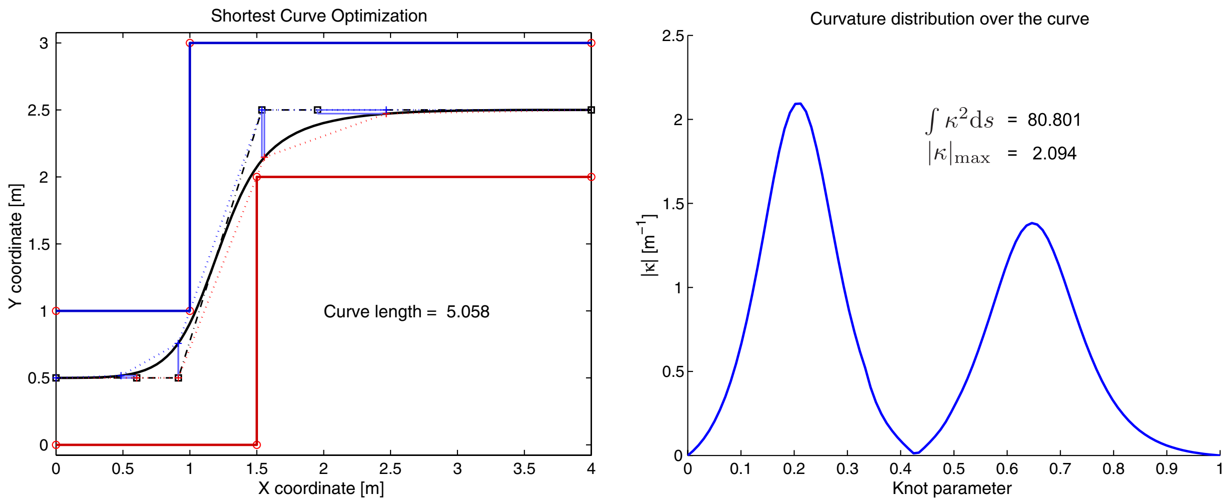
Suppose the world environment is decomposed into uniform cells consisting of square cells $c_{k,\ell}$ of size $2h \times 2h$. A four-connectivity relationship between neighboring cells is henceforth adopted for convenience. A high-level path planner computes an optimal path as a sequence of cells from the current cell to the goal cell. The resulting path sequence can be expressed as a pathword, by which transitions toward the north, south, east, and west directions between two cells are encoded by N, S, E, and W, respectively. One thus specifies the range of interest within a four-cell horizon from the current cell, as shown in Fig. 7. If the goal cell is located inside the horizon, we are finished. If the goal set is outside the horizon, note that any path sequence must necessarily pass through one of the cells at the horizon boundary to reach the goal. Let a local path sequence from the current cell to any boundary cell be a local path instance. If the path is supposed to visit each cell only once, the number of all possible combinations of local path instances is finite. In addition, by taking advantage of the symmetry about the x axis (east direction) and y axis (north direction), one needs to investigate only local path instances restricted to one quadrant of the original 7×7 cell grid.

Without loss of generality, one may thus consider local path instances only inside the first quadrant, as shown in Fig. 8. Furthermore, by taking advantage of the symmetry about the diagonal axis, one may only consider local path templates that start from the current cell and end at one of the top boundary cells $c_{0,3}$, $c_{1,3}$, $c_{2,3}$, and $c_{3,3}$. By applying the symmetric operations along the horizontal, vertical, and diagonal axes, any local path instances can be deduced from these path templates. To find all possible combinations of path sequences to reach these cells, one must describe the necessary path rules to determine a unique local path template. These are listed as follows:

1) For the terminal conditions, suppose a local path instance is restricted inside the first quadrant; that is, it never goes outside the horizon before it reaches a terminal cell on the top boundary. The terminal cell should be one of the top boundary cells, except the cell $c_{3,3}$. The reason for this is attributed to the four connectivity between cells. A path instance that reaches $c_{3,3}$ necessarily passes through one of the adjacent boundary cells, either $c_{2,3}$ or $c_{3,2}$. Thus, one regards the path instances to the cell $c_{3,3}$ as a subset of the path instances to $c_{2,3}$ and excludes those from further consideration. To come up with a path sequence that reaches one of the boundary cells, a corresponding pathword should have a certain number of occurrences of N, S, E, and



a) Smooth curve in the channel



b) Shortest curve in the channel

Fig. 6 Two optimization results.

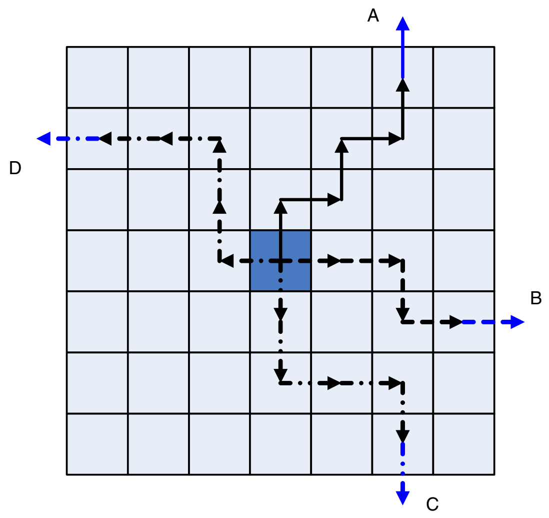


Fig. 7 Examples of path sequences starting from the current cell at the center. We adopt four connectivity between cells. The goal cell is located beyond the horizon. Possible path sequences are the path A, written as NENEN . . . ; the path B, written as EESE . . . ; the path C, written as SSEES . . . ; and the path D, written as WNNWW . . . .

W. For example, in order to reach the top boundary cells, the number of occurrence of N minus the number of occurrence of S is three when a four-cell horizon is considered.

2) For the self-avoiding path, the path must visit each cell exactly once, never intersecting itself. From this rule, one explicitly prevents pathological cases such as a cyclic loop in the path templates. This type of path can be described by a self-avoiding walk [24] on a square-cell grid. The total number of self-avoiding walks in an $m \times n$ grid, which starts from a corner and ends at the opposite corner by only horizontal and vertical steps, is computed using a recurrence relation [24]. It follows that the candidates of self-avoiding paths from the current cell $c_{0,0}$ to the top boundary cells are chosen from the self-avoiding walk on a $m \times 3$ grid, where $m = 0, 1, 2$. Among these candidates, only a certain number of self-avoiding paths will be considered in the path templates.

3) For path optimality, the higher-level path-planning algorithm provides the optimal cell sequence to be followed. The optimal cell sequence is calculated so as to minimize the accumulated transition cost from the current cell to the goal cell. Typically, a directed edge cost is assigned to each transition to N, S, E, and W cells, taking into account the cost associated with the cells as follows:

$$\mathcal{J}(u, v) = f(v) + \alpha g(u, v) \tag{17}$$

where $f(v)$ is a positive obstacle cost associated with the target cell v, $g(u, v)$ is the Manhattan distance cost between u and v, and $\alpha \geq 0$ is a weight constant. Consider, for example, the path corresponding to the pathword ENW, which represents a transition among four cells u, v, w, and z in this order. The accumulated cost for this transition is computed by

$$\mathcal{J}(u,v) + \mathcal{J}(v,w) + \mathcal{J}(w,z) > \mathcal{J}(u,z) \tag{18}$$

which turns out to be greater than the direct transition cost from u to z by a single path sequence N. It follows that the transition ENW is not optimal and neither are ESW, NES, NWS and the other remaining pathwords. Consequently, we disregard any nonoptimal cell sequence when investigating the candidates of self-avoiding paths.

When establishing the preceding path rules, it was assumed that each local path template necessarily ends up at one of the top boundary cells. In certain cases, however, the path sequence may be given in such a way that it crosses the quadrant boundary. In other words, the path sequence that starts from the current cell at the center of grid is composed of cells belonging to more than one quadrant. If this is the case, one can infer that the path sequence will finally exit the finite horizon after passing through at least two quadrants, which makes it difficult to take advantage of the symmetry of the templates. In particular, if one wants to consider all possibilities of interquadrant transitions, the number of templates will increase, thus losing the benefit of small-sized templates. To retain the symmetry of the templates, one considers cells between the quadrants as additional terminal cells. Applying the path rules to a 7×7 cell grid (thus a 4×4 cell grid for the first quadrant), only the cell $c_{0,2}$ (see Fig. 8) can be considered as an additional terminal cell. The other cells on the y axis cannot be terminal cells because the local path instances reaching them would conflict with the path rules discussed in the preceding paragraphs. Any local path instance starting from the center cell to $c_{0,2}$, satisfying the path rules, is appended to the path templates.

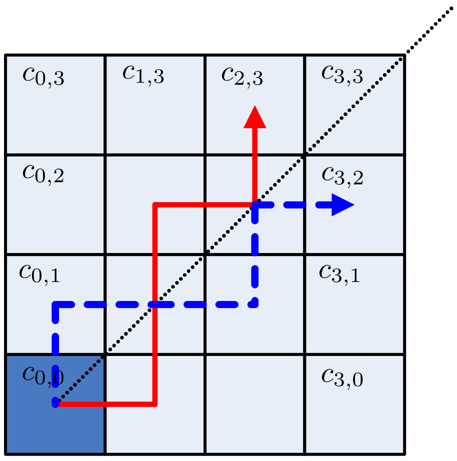


Fig. 8 Local path instances in the first quadrant. From the additional symmetry about the diagonal axis, it is possible to transform the path instance drawn by a dashed line (NEENE) to the path instance drawn by a solid line (ENNEN). Path rules are given to determine several unique path instances to reach the cell at the top boundary.

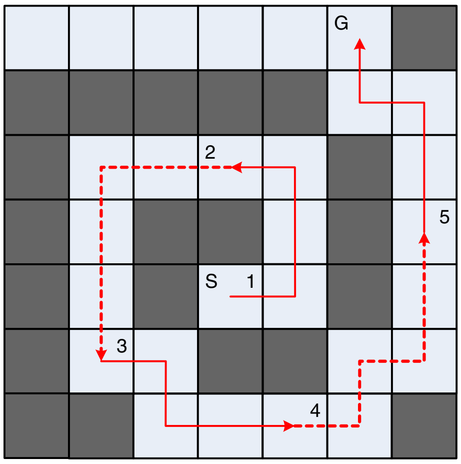
Following the preceding path rules, one finally ends up with the path templates for the local path instances for the first quadrant. These are summarized in Table 1. Figure 9 shows an example of using the path templates for a given path sequence. The starting cell is located at the center, whereas the path sequence was computed by avoiding the shaded obstacle cells. For this example, in order to reach the goal cell, five local path instances are required to represent the overall path sequence. To search for the path templates, one first gets the actual pathwords that correspond to each of the local path instances. Then, one associates these pathwords to the first quadrant by using the required symmetry operations, that is, the horizontal (H), vertical (V), or diagonal (D) reflections. One can then identify the corresponding entry in the path templates. The table in Fig. 9 illustrates that five different path templates are used in order to construct the composite path from the starting cell to the goal cell for the example shown.

Table 1 Path templates for local path instances on the first quadrant

Destination cell	Pathwords
$c_{0,3}$	NNN, ENNWN, EENNWWN
$c_{1,3}$	NNEN, ENNN, EENNWN
$c_{2,3}$	NNEEN, NENEN, ENNEN, ENENN, EENNN
$c_{0,2}$	ENNW, EENNWW

B. Construction of B-Spline Path Templates

The B-spline path templates consist of a set of B-spline curves inside the corresponding channels. A channel corresponding to an optimal path sequence over a finite planning horizon is determined in a manner such that the outmost border lines of each cell yield a channel polygon. The channel polygon is then divided by two left and right polylines, which serve as channel constrains during optimization. For the sake of convenience, the boundary conditions for each B-spline curve are imposed such that the B-spline curve starts from the center of the first cell of the local path instance and ends at the center of the last cell of the local path instance. The tangent directions at each end of the curve are imposed such that they direct toward the center of the next adjacent cell, whereas the curvature values are all set to zero. Subsequently, the optimization problem discussed in Sec. III is solved using the composite cost Eq. (14) or (15), subject to a set of channel constraints and the associated boundary conditions. Figure 10 shows the results of this optimization for the B-spline path templates corresponding to each local path instance shown in Table 1.



Path #	Pathwords	Templates	Operations
1	ENNW	ENNW	-
2	WWSSS	EENNN	H, V
3	ESEE	NNEN	V, D
4	ENENN	ENENN	-
5	NNWN	NNEN	V

Fig. 9 Example incorporating the path templates on a complex path sequence. Five local path instances are connected to one another in order to reach the goal cell. The actual pathwords are recovered from the path templates with the corresponding symmetry operations of the H, V, or D reflections.

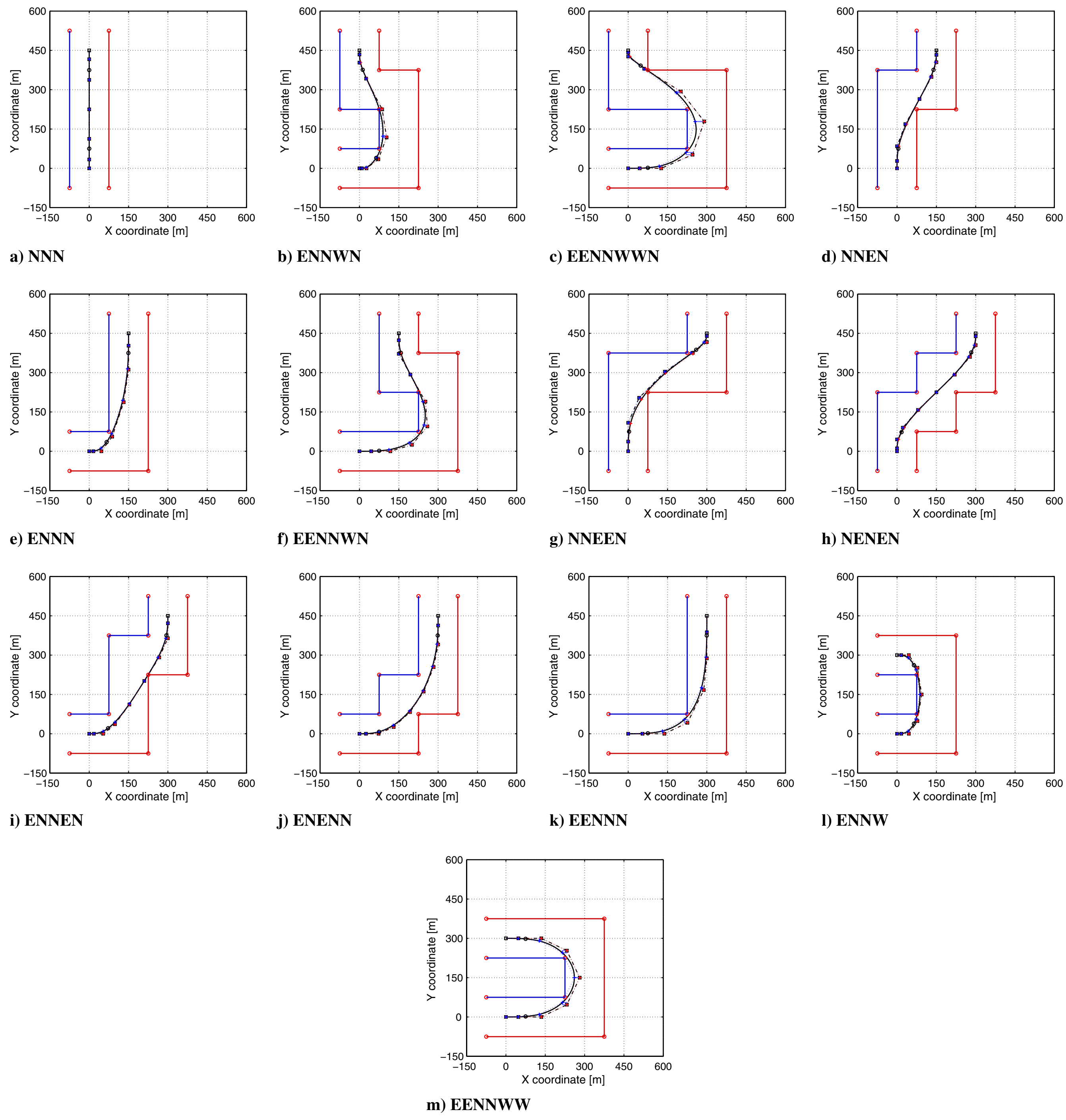


Fig. 10 B-spline path templates from the channel optimization results. Each plot corresponds to the local path instance in Table 1.

V. On-Line Path-Smoothing Algorithm

The proposed path-smoothing algorithm computes a composite B-spline curve that smooths the discrete path sequence obtained by the high-level path planner. When combining the B-spline curves in the templates into a composite B-spline curve, it is necessary to keep the composite B-spline curve smooth, especially at the junctions of two adjacent B-spline curves. This is achieved by employing a B-spline merging technique. By merging a B-spline path template into the existing B-spline path curve, it is shown that a single, non-fragmented B-spline path can be computed. Thus, one avoids using transient B-splines and a complicated switching logic, as was the case in [25]. Additionally, large curvature variations resulting from the use of transient B-splines are effectively eliminated, thus resulting in a smoother path.

A. Approximate Merging of B-Spline Path Segments

Approximate merging of two B-spline curves is the process of combining two, or more, B-spline curves into a single B-spline curve, for which the shape approximates the original B-spline curves as close as possible. It is assumed that the B-splines to be merged have the same degree. During the merging process it is necessary to recalculate some control points to alter the shape of the curves. It is also necessary to reparameterize each knot vector to be consistent with the newly computed control points. To accomplish the approximate merging, we adopt the merging algorithm proposed by Tai et al. [26] along with some additional conditions in order to accommodate the channel constraints introduced in Sec. III.A.

Let $\boldsymbol{P}(u)$ and $\boldsymbol{Q}(v)$ be two B-splines of the same degree p, which are adjacent to each other. Let the knot vectors be denoted by

$\boldsymbol{U} = \{u_0, u_1, \ldots, u_{n_1+p}\}$ and $\boldsymbol{V} = \{v_0, v_1, \ldots, v_{n_2+p}\}$ and let the control points be denoted by $\boldsymbol{P}_i (i = 0, \ldots, n_1)$ and $\boldsymbol{Q}_i (i = 0, \ldots, n_2)$, respectively. Without loss of generality, the two B-splines are clamped at each endpoint with knot multiplicity $(p + 1)$. Let $\boldsymbol{R}(w)$ be the merged B-spline of degree p having control points $\boldsymbol{R}_i (i = 0, 1, \ldots, n_1 + n_2 - p + 1)$ and a knot vector $\boldsymbol{W} = \{w_0, w_1, \ldots, w_{n_1+n_2+2}\}$. One needs to ensure that the two B-splines share common derivatives up to degree $(p - 1)$ at a certain merging knot w_m (precise merging condition). It follows that curvature continuity is thus automatically satisfied for a cubic B-spline, and continuity of the derivative of the curvature is satisfied for a quartic B-spline. Later on, we explain how to determine the merging knot parameter w_m in conjunction with the knot vectors of the original B-spline curves. For now, the merging knot is assumed to be chosen by the middle knot of the combined knot vector of the two splines. Subsequently, the precise merging condition is given as follows:

$$\boldsymbol{P}^{(k)}(w_m) = \boldsymbol{Q}^{(k)}(w_m), \qquad k = 0, \ldots, p - 1 \tag{19}$$

Using the basis function of the corresponding B-spline curves, Eq. (19) can be arranged in terms of the control points of the kth derivative of the B-splines, $\boldsymbol{P}_i^k$ and $\boldsymbol{Q}_i^k$ $(k \le p - 1)$, given by

$$\sum_{i=n_1-p}^{n_1-k} N_{i,p-k}^{\boldsymbol{U}}(w_m)\boldsymbol{P}_i^k = \sum_{i=0}^{p-k} N_{i,p-k}^{\boldsymbol{V}}(w_m)\boldsymbol{Q}_i^k, \quad 0 \le k \le p - 1 \tag{20}$$

where $N_{i,p-k}^{\boldsymbol{U}}$ and $N_{i,k}^{\boldsymbol{V}}$ are the B-spline basis functions corresponding to each knot vector $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively, where $\boldsymbol{U} \triangleq \{0, \ldots, 0, u_{p+1}, \ldots, u_{m_1-p-1}, 1, \ldots, 1\}$ and $\boldsymbol{V} \triangleq \{0, \ldots, 0, v_{p+1}, \ldots, v_{m_2-p-1}, 1, \ldots, 1\}$, where $0 \le k \le p - 1$. The control points of the kth derivatives of the curves $\boldsymbol{P}_i^k$ $(0 \le k \le p - 1)$ can be computed recursively in terms of the control points $\boldsymbol{P}_i$ as follows [18]:

$$\boldsymbol{P}_i^k = \begin{cases} \boldsymbol{P}_i, & k = 0; \\ \frac{p-k+1}{u_{i+p+1} - u_{i+k}}(\boldsymbol{P}_{i+1}^{k-1} - \boldsymbol{P}_i^{k-1}), & k > 0 \end{cases} \tag{21}$$

It should be noted that because Eq. (21) is linear in terms of $\boldsymbol{P}_i$, it can be rearranged as $\boldsymbol{P}_i^k = \sum_{j=n_1-p}^{n_1} a_{ij}^{(k)}\boldsymbol{P}_j$, where $[a_{ij}^{(k)}]$ is a matrix of dimensions $(p + 1 - k) \times (p + 1)$ where $k \le p - 1$. Similarly, $\boldsymbol{Q}_i^k$ can be rearranged as $\boldsymbol{Q}_i^k = \sum_{j=0}^{p} b_{ij}^{(k)}\boldsymbol{Q}_j$. Using these equalities, Eq. (20) can be rewritten in the following form:

$$\sum_{j=n_1-p}^{n_1}\left(\sum_{i=n_1-p}^{n_1-k} a_{ij}^{(k)} N_{i,p-k}^{\boldsymbol{U}}(u_{w_m})\right)\boldsymbol{P}_j - \sum_{j=0}^{p}\left(\sum_{i=0}^{p-k} b_{ij}^{(k)} N_{i,p-k}^{\boldsymbol{V}}(w_m)\right)\boldsymbol{Q}_j = 0 \tag{22}$$

To merge two B-splines $\boldsymbol{P}(u)$ and $\boldsymbol{Q}(v)$, their control points are modified so that these are precisely merged. In other words, by taking into account the linear independency of the B-spline basis functions and the number of equations in Eq. (22), one perturbs $(p + 1)$ control points of each curve in order to impose the precise merging condition. To this end, let $\boldsymbol{\epsilon}_i$, $(i = n_1 - p, \ldots, n_1)$ and δ_i, $(i = 0, \ldots, p)$ be perturbations from the original control points of $\boldsymbol{P}_i$ and $\boldsymbol{Q}_i$, respectively. By incorporating these perturbations in Eq. (22), one obtains,

$$\sum_{j=n_1-p}^{n_1} A_j^{(k)}(\boldsymbol{P}_j + \boldsymbol{\epsilon}_j) - \sum_{j=0}^{p} B_j^{(k)}(\boldsymbol{Q}_j + \delta_j) = 0 \tag{23}$$

where the symbols $A_j^{(k)}$ and $B_j^{(k)}$ have been used instead of $\sum_{i=n_1-p}^{n_1-k} a_{ij}^{(k)} N_{i+1,p-k}^{\boldsymbol{U}}(w_m)$ and $\sum_{i=0}^{p} b_{ij}^{(k)} N_{i+1,p-k}^{\boldsymbol{V}}(w_m)$, respectively.

In addition to the precise merging condition, the channel constraints presented in Sec. III.A are also imposed in order to ensure obstacle avoidance. For instance, suppose that two B-spline templates overlap at a single cell (see Fig. 9). The point on the merged curve corresponding to the merging knot must be contained within the boundary of the merging cell. That is, the following inequality constraint is imposed:

$$\left\|\hat{\boldsymbol{P}}(w_m) - \boldsymbol{P}_c\right\|_\infty = \left\|\sum_{i=n_1-p}^{n_1}(\boldsymbol{P}_i + \boldsymbol{\epsilon}_i)N_{i,p}^{\boldsymbol{U}}(w_m) - \boldsymbol{P}_c\right\|_\infty \le h \tag{24}$$

where $\|\cdot\|_\infty = \max(|x_1|, |x_2|)$ and $\boldsymbol{P}_c$ is the center of the square cell for which the size is $2h$.

The performance index is given in a quadratic form and minimizes the perturbation so that the merged curve approximates the original curve:

$$J(\{\boldsymbol{\epsilon}_j\}_{j=n_1-p}^{n_1}, \{\delta_j\}_{j=0}^{p}) = \sum_{j=n_1-p}^{n_1}(\boldsymbol{\epsilon}_j)^\top(\boldsymbol{\epsilon}_j) + \sum_{j=0}^{p}(\delta_j)^\top(\delta_j) \tag{25}$$

By solving this quadratic optimization problem subject to the equality constraints (23) and the inequality constraint (24), one obtains $(p + 1)$ modified control points for each of the original B-spline curves so that $\hat{\boldsymbol{P}}_i \triangleq \boldsymbol{P}_i + \boldsymbol{\epsilon}_i$ and $\hat{\boldsymbol{Q}}_j \triangleq \boldsymbol{Q}_j + \boldsymbol{\delta}_j$.

After obtaining the modified control points, the second step of merging involves the reparameterization of the two knot vectors to turn them into a single knot vector that is compatible with the newly computed control points. Given the knot vector $\boldsymbol{U}$ of the first B-spline, an affine transformation on the second knot vector $\boldsymbol{V}$ is first performed such that $v_0' \ge u_{n_1+p}$ and $\boldsymbol{v}_i'$ are nondecreasing. It should be mentioned that the affine transformation of the knot vector has no effect on the B-spline. On the other hand, recall that $(p + 1)$ control points of each B-spline are involved in merging; it follows that these control points can be shared in common for the merged B-spline curve. Consequently, when the two knot vectors are merged it is required to take into account the number of these shared control points. To this end, let the merging knot w_m be chosen among the knots of $\boldsymbol{U}$ and $\boldsymbol{V}'$ such as

$$U = \{u_0, u_1, \ldots, u_{n1}, u_{n_1+1} = w_m, \ldots, u_{n_1+p}\} \tag{26}$$

and

$$\boldsymbol{V} = \{v_0', v_1', \ldots, v_p' = w_m, v_{p+1}', \ldots, v_{n_2+p}'\} \tag{27}$$

where the prime denotes the corresponding knot vectors after the affine transformation on $\boldsymbol{V}$. Next, a reparameterization of the knot vectors is applied to remove any multiplicities. This can be done, for instance, by the knot adjustment algorithm presented in [26]. For the sake of simplicity, denote by $\hat{\boldsymbol{P}}_i$ $(i = 0, 1, \ldots, n_1)$ and $\hat{\boldsymbol{Q}}_i$, $(i = 0, 1, \ldots, n_2)$ the control points after the knot adjustments followed by perturbing the original control points. The composite knot vector is constructed from Eqs. (26) and (27) as follows:

$$\boldsymbol{W} = \{u_0, u_1, \ldots, u_{n_1}, u_{n_1+1} = w_m = v_p', v_{p+1}', \ldots, v_{n_2+p}'\} \tag{28}$$

and the control points of the merged B-spline are

$$\boldsymbol{R} = \{\widehat{\boldsymbol{P}}_0, \widehat{\boldsymbol{P}}_1, \ldots \widehat{\boldsymbol{P}}_{n_1-p+1} = \widehat{\boldsymbol{Q}}_0, \ldots, \widehat{\boldsymbol{P}}_{n_1-1} = \widehat{\boldsymbol{Q}}_p, \ldots, \widehat{\boldsymbol{Q}}_m\} \tag{29}$$

B. Path Generation by Merging B-Spline Path Templates

Assuming that the high-level path planner provides an accurate path sequence of obstacle-free cells within the planning horizon from the current location of the UAV, the online path generation is carried out by merging the existing B-spline path with the newly computed B-spline segment that corresponds to the path sequence within the finite horizon. As the UAV reaches the planning horizon, the process is repeated again until the UAV finally arrives at its destination.

The reference path for the path-following controller is represented by a single B-spline curve using the B-spline merging algorithm, which is parameterized by a nondecreasing knot vector as the new B-spline template is merged. Subsequently, with a given consistent parameterization over the entire path, the execution of the path-following controller can be simplified. One may want to compare this approach to the previous B-spline stitching method of [25], in which a complicated switching logic was employed. The smoothness of the B-spline path is automatically satisfied with the use of the proposed B-spline merging algorithm. On the other hand, in [25], path smoothness was ensured by the introduction of a transient B-spline such that continuity with respect to position, tangent angle, and curvature is imposed at each end of the transient B-spline. One drawback of the stitching method of [25] is that stitching two B-spline templates yields a transient B-spline curve of perhaps large curvature variation for 90 deg turns at the stitching cell. This is because the transient B-spline is relatively short, but the turn needs to take place rapidly, resulting in a steep increase of the curvature value. In contrast, the merging algorithm proposed in this paper results in a smaller curvature variation even for 90 deg turns. This can be attributed to the property of lowest torsional energy of a B-spline curve; hence, the curvature variation is uniformly distributed over the whole B-spline. Subsequently, the path generation via merging can lower the curvature variation.

It should be noted that the maximum curvature value of the path curve is imposed by the maneuverability limits of the vehicle. For a fixed-wing UAV, for example, these are determined by the speed, lift-to-weight ratio, and other similar factors. In this paper, the maneuverability limits are implicitly captured by the imposed constraints on the curvature during optimization; in our approach, curvature thus serves as a geometric surrogate of the dynamics. In addition, during the offline optimization step to calculate the path templates, the curvature constraint is included as a soft constraint. The path templates are individually examined after the optimization to verify that the actual curvature value is kept within bounds of maneuverability limits of the UAV. By incorporating the B-spline merging algorithm, the proposed approach can be applied to efficiently generate a flyable path that is compatible to the maneuverability limits of the UAV.

In the following, an illustrative example is presented to compare the proposed B-spline merging algorithm to the B-spline stitching algorithm of [25]. For the sake of convenience, it is assumed that a discrete path is given as a sequence of square cells with four connectivity as shown in Fig. 11a. The B-spline path templates proposed in Sec. IV are incorporated in order to generate a smooth path curve. Starting from the cell's top-left corner, there exist 14 path instances that are connected to one another around the middle cells. The path stitching between two B-spline templates is achieved by placing a transient B-spline curve in the middle cell. The transient B-spline is obtained in such a manner that it replaces a portion of two adjacent B-spline curves, while imposing continuity at the stitching points. However, a 90 deg turn at the stitching cell may lead to a transient B-spline with large curvature variation, which reduces the overall smoothness of the composite path. The smoothing result from the B-spline merging algorithm is shown in Fig. 11a. As discussed in the preceding section, the entire path is represented by a single B-spline curve, and is likely to yield a smoother path than the one obtained by the stitching algorithm.

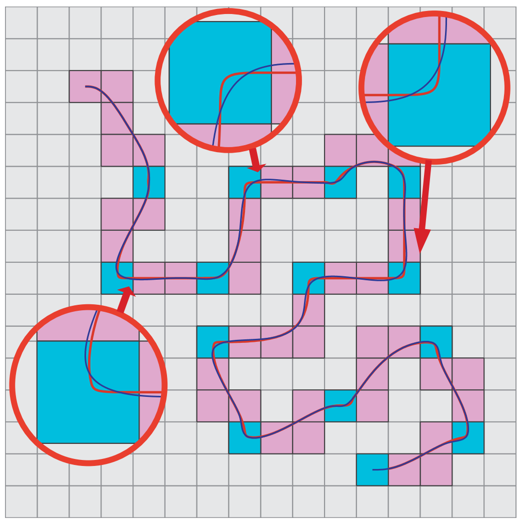
To confirm these claims, Fig. 11b shows a comparison of the curvature variation along the path. The curvature peaks occur when the discrete path turns 90 deg. The stitching algorithm results in high curvature variation during the turn, whereas the path generation using B-spline merging algorithm effectively reduces the curvature variation, thus increasing the smoothness of the resulting B-spline path.

VI. Simulation Results

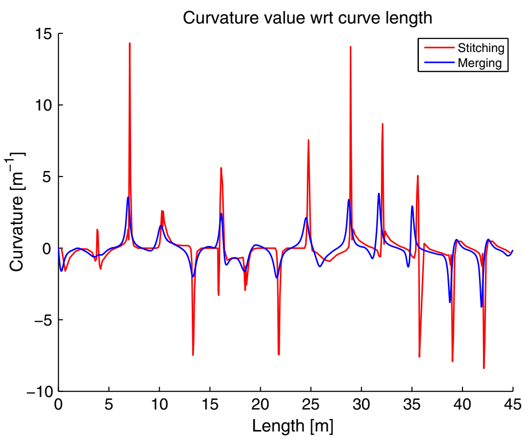
The proposed algorithm has been implemented in conjunction with a high-level discrete planner, which provides a sequence of obstacle-free cells in a unknown environment. In our implementation, the $\mathcal{D}^*$-lite algorithm was used for this purpose, as it allows efficient replanning. The $\mathcal{D}^*$-lite (Dynamic $\mathcal{A}^*$-lite) algorithm was proposed by Koenig and Likhachev [7] for path planning in unknown or partially known environments. The algorithm reuses information from the previous search to find the solution at the next iteration much faster than solving each iteration from scratch.

It was assumed that the UAV navigates over an unknown environment, while updating the map with information gathered using a suitable proximity sensor. The world topographic data are given as a 256×256 units (pixels) map covering an area of approximately 4.8 km^2. A uniform cell decomposition of cell size 8×8 pixels was adopted. The range of the proximity sensor is chosen to be $r = 28$, thus resulting in the finite horizon window by a 7×7 square cell grids.

The results from the on-line path-smoothing algorithm combined with the $\mathcal{D}^*$-lite path planning are shown in Fig. 12. Specifically, Fig. 12 shows the evolution of the path at different time steps, as the agent moves to the final destination. At each step, the best proposed path is drawn by a dashed-dot line, and the actual path generated by the algorithm is drawn by a solid line. A local channel is drawn by thin



a) Path smoothing



b) Curvature

Fig. 11 Example of stitching the two B-spline curves with a transient cubic B-spline curve. The dashed boxes represent overlapping cells of two successive path templates where the corresponding transient B-spline curve is placed.

polylines, which corresponds to the discrete path sequence from the $\mathcal{D}^*$-lite algorithm. Accordingly, a smooth path segment to be followed is obtained from the B-spline path templates. Whenever the $\mathcal{D}^*$-lite algorithm updates with a (possibly new) path sequence, as the agent approaches close to the end of each path segment, the on-line path-smoothing algorithm proceeds to merge the existing B-spline path with the newly obtained path segment within the finite horizon. This process repeats until the UAV reaches the final destination, as shown in Fig. 12c.

The main benefit of the proposed navigation algorithm is that it is relatively "cheap" in terms of use of computational resources, thus making it suitable for embedded implementation in UAVs. The proposed algorithm uses small computer memory because the B-spline path templates are stored using only a small number of control points and knot sequences. Furthermore, by incorporating path templates over a finite horizon that have been computed offline, the proposed algorithm generates a smooth path quickly, while ensuring the smoothness of the entire path.

It should be noted that the proposed merging algorithm in Sec. V.A requires the solution of a quadratic programming, which has to be solved on-line. Although this increases the overall on-line computational overhead somewhat, the size of optimization problem is always fixed regardless of the size of the B-spline path. Recalling that the quadratic programming can be solved in polynomial time [27], the proposed algorithm can thus be tailored to fit the onboard computational resources of the UAV. An experimental implementation of the complete hierarchical path-planning scheme on a small UAV 50 MHz microcontroller [28,29] showed that the proposed path-generation algorithm takes about 5 ms to compute the path segments from the path templates out of a 20 ms sampling interval, enabling hard real-time implementation. The results from these tests can be found in [30,31].

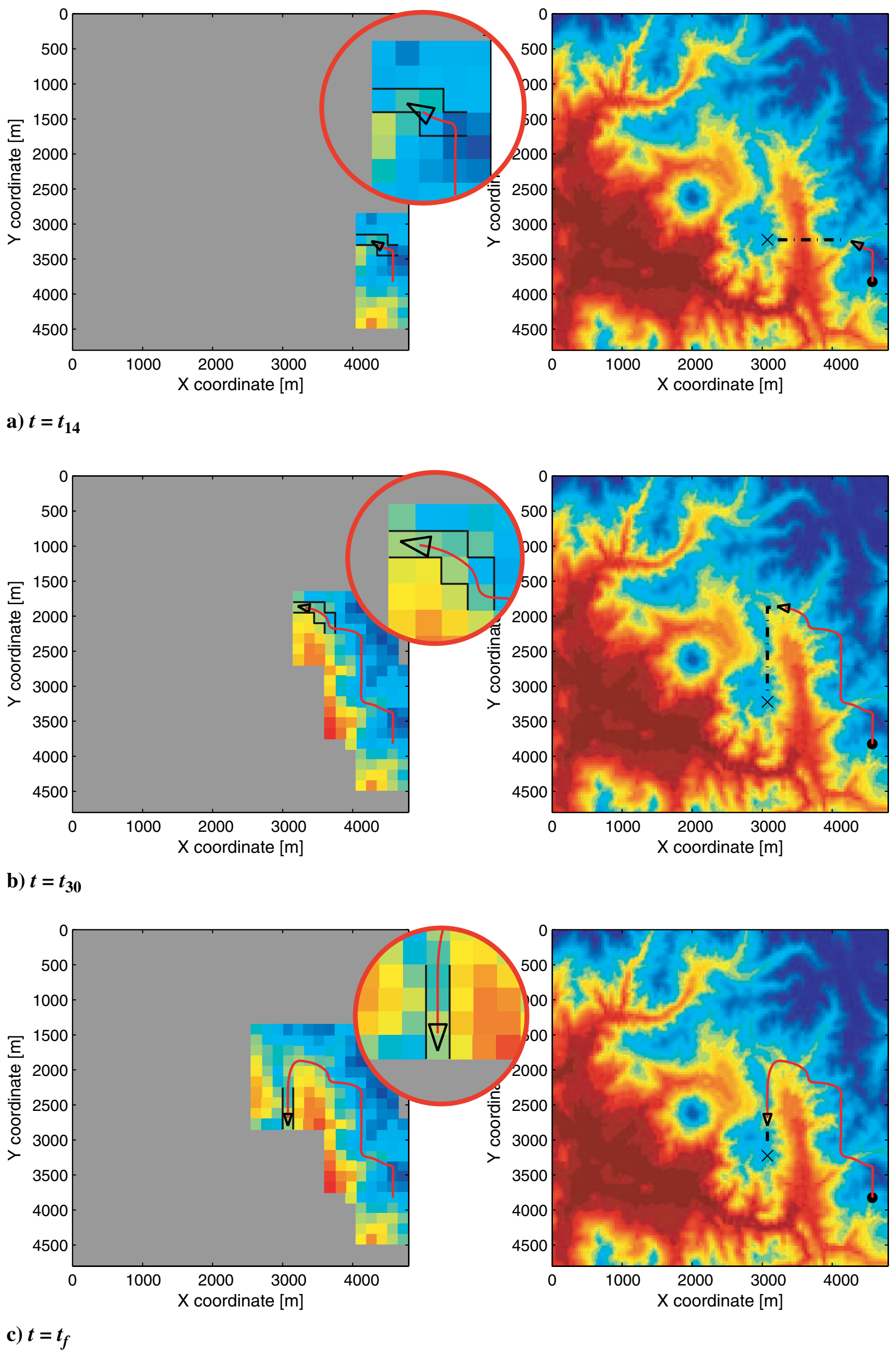


Fig. 12 On-line path smoothing in conjunction with replanning using the $\mathcal{D}^*$-lite algorithm. Dashed-dot lines represent the currently tentative optimal path obtained from the $\mathcal{D}^*$-lite algorithm, based on the distance cost outside the finite horizon window. Actual reference path to be followed by the agent using the B-spline path templates is represented by a solid line.

VII. Conclusions

In this paper, a new on-line path-smoothing algorithm that incorporates path templates for generating a path derived from a high-level path planner, is presented. The path templates are composed of a set of B-spline curves, obtained from an offline optimization step such that each path instance stays inside the prescribed channel, hence avoiding obstacles. Owing to the incremental implementation, which involves merging B-spline path templates in conjunction with a high-level path planner, the proposed approach has the benefit of generating a reference path that is appropriate for a complex environment with obstacles, while preserving the smoothness of the composite curve. When used in conjunction with a high-level path planner (such as $\mathcal{D}^*$-lite) the algorithm yields close-to-optimal paths in a changing environment. Simulation results validate the effectiveness of the proposed algorithm. The algorithm provides a complete solution to the obstacle-free path-generation problem and is especially suitable for real-time implementation for small size UAVs having limited computational resources.

Acknowledgments

Partial support for this work has been provided by US Army Research Office Multidisciplinary University Research Initiative (MURI) award W911NF-11-0046 and National Science Foundation awards CMS-0510259 and CMMI-0856565.

References

[1] McLain, T., Chandler, P., and Pachter, M., "A Decomposition Strategy for Optimal Coordination of Unmanned Air Vehicles," *Proceedings of the American Control Conference*, Chicago, IL, IEEE, 2000, pp. 369–373.

[2] Beard, R. W., McLain, T. W., Goodrich, M., and Anderson, E. P., "Coordinated Target Assignment and Intercept for Unmanned Air Vehicles," *IEEE Transactions on Robotics and Automation*, Vol. 18, 2002, pp. 911–922.
doi:10.1109/TRA.2002.805653

[3] McLain, T. W., and Beard, R. W., "Coordination Variables, Coordination Functions, and Cooperative Timing Missions," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 1, 2005, pp. 150–161.
doi:10.2514/1.5791

[4] Kanayama, Y., and Hartman, B. I., "Smooth Local Path Planning for Autonomous Vehicles," *Proceedings of IEEE International Conference on Robotics and Automation*, Vol. 3, IEEE, May 1989, pp. 1265–1270.

[5] Scheuer, A., and Laugier, C., "Planning Sub-Optimal and Continuous-Curvature Paths for Car-Like Robots," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Victoria, Canada, IEEE, Oct. 1998, pp. 25–31.

[6] Anderson, E. P., Beard, R. W., and McLain, T. W., "Real-Time Dynamic Trajectory Smoothing for Unmanned Air Vehicles," *IEEE Transactions on Control Systems Technology*, Vol. 13, No. 3, 2005, pp. 471–477.
doi:10.1109/TCST.2004.839555

[7] Koenig, S., and Likhachev, M., "$\mathcal{D}^*$ Lite," *Proceedings of the National Conference of Artificial Intelligence*, AAAI Press, Menlo Park, CA, 2002, pp. 476–483.

[8] Judd, K. B., and McLain, T. W., "Spline Based Path Planning for Unmanned Air Vehicles," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA Paper 2001-4238, Aug. 2001.

[9] Vázquez, G. B., Sossa, A. J. H., and Díaz-de-León S., J. L., "Auto Guided Vehicle Control Using Expanded Time B-Splines," *IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 3, San Antonio, TX, IEEE, Oct. 1994, pp. 2786–2791.

[10] Berglund, T., Jonsson, H., and Söderkvist, I., "An Obstacle-Avoiding Minimum Variation B-Spline Problem," *Proceedings of 2003 International Conference on Geometric Modeling and Graphics*, IEEE, July 2003, pp. 156–161.

[11] Dyllong, E., and Visioli, A., "Planning and Real-Time Modifications of a Trajectory Using Spline Techniques," *Robotica*, Vol. 21, No. 5, 2003, pp. 475–482.
doi:10.1017/S0263574703005009

[12] Maekawa, T., Noda, T., Tamura, S., Ozaki, T., and Machida, K., "Curvature Continuous Path Generation for Autonomous Vehicle using B-Spline Curves," *Computer-Aided Design*, Vol. 42, No. 4, 2010, pp. 350–359.
doi:10.1016/j.cad.2009.12.007

[13] Choi, J.-W., Curry, R. E., and Elkaim, G. H., "Continuous Curvature Path Generation Based on Bézier Curves for Autonomous Vehicles," *IAENG International Journal of Applied Mathematics*, Vol. 40, No. 2, 2010, pp. 91–101.

[14] Sprunk, C., Lau, B., Pfaffz, B., and Burgard, W., "Online Generation of Kinodynamic Trajectories for Non-Circular Omnidirectional Robots," *IEEE International Conference on Robotics and Automation*, Shanghai, China, IEEE, May 2011, pp. 72–77.

[15] Lutterkort, D., and Peters, J., "Smooth Paths in a Polygonal Channel," *Proceedings of the Fifteenth Annual Symposium on Computational Geometry*, Miami Beach, FL, Association for Computing Machinery (ACM), New York, NY, 1999, pp. 316–321.

[16] Lutterkort, D., and Peters, J., "Tight Linear Envelopes for Splines," *Numerische Mathematik*, Vol. 89, No. 4, 2001, pp. 735–748.
doi:10.1007/s002110100181

[17] Nairn, D., Peters, J., and Lutterkort, D., "Sharp, Quantitative Bounds on the Distance Between a Polynomial Piece and its Bézier Control Polygon," *Computer Aided Geometric Design*, Vol. 16, No. 7, 1999, pp. 613–631.
doi:10.1016/S0167-8396(99)00026-6

[18] Piegl, L., and Tiller, W., *The NURBS Book–Monographs in Visual Communication*, 2nd ed., Springer–Verlag, Berlin, 1997, pp. 47–116.

[19] Frain, G., *Curves and Surfaces for CAGD — A Practical Guide*, 5th ed., Morgan Kaufmann, San Mateo, CA, 2001, pp. 119–146.

[20] Lutterkort, D., and Peters, J., "The Distance Between a Uniform (B-) Spline and Its Control Polygon," Dept. of Computer and Information Science and Engineering, Univ. of Florida, TR-98-013, Gainesville, FL, Sept. 1998.

[21] *Optimization Toolbox™ User's Guide*, 7th ed., The Mathworks, Inc., Natick, MA, 2003, pp. 3–33.

[22] Jung, D., and Tsiotras, P., "Multiresolution On-Line Path Planning for Small Unmanned Aerial Vehicles," *Proceedings of the American Control Conference*, Seattle, WA, IEEE, June 2008, pp. 2744–2749.

[23] Tsiotras, P., Jung, D., and Bakolas, E., "Multiresolution Hierarchical Path-Planning for Small UAVs Using Wavelet Decompositions," *Journal of Intelligent and Robotic Systems*, Vol. 66, No. 4, 2012, pp. 505–522.
doi:10.1007/s10846-011-9631-z

[24] Finch, S. R., *Mathematical Constants*, Cambridge Univ. Press, Cambridge, England, U.K., 2003, pp. 331–339.

[25] Jung, D., and Tsiotras, P., "On-Line Path Generation for Small Unmanned Aerial Vehicles Using B-Spline Path Templates," *AIAA Guidance, Navigation and Control Conference*, AIAA Paper 2008-7135, 2008.

[26] Tai, C.-L., Hu, S.-M., and Huang, Q.-X., "Approximate Merging of B-spline Curves via Knot Adjustment and Constrained Optimization," *Computer-Aided Design*, Vol. 35, No. 10, 2003, pp. 893–899.
doi:10.1016/S0010-4485(02)00176-8

[27] Kozlov, M. K., Tarasov, S. P., and Khachiyan, L. G., "The Polynomial Solvability of Convex Quadratic Programming," *USSR Computational Mathematics and Mathematical Physics*, Vol. 20, No. 5, 1980, pp. 223–228.
doi:10.1016/0041-5553(80)90098-1

[28] Jung, D., and Tsiotras, P., "Inertial Attitude and Position Reference System Development for a Small UAV," *AIAA Infotech at Aerospace*, AIAA Paper 07-2768, May 2007.

[29] Jung, D., and Tsiotras, P., "Modelling and Hardware-in-the-Loop Simulation for a Small Unmanned Aerial Vehicle," *AIAA Infotech at Aerospace*, AIAA Paper 07-2763, May 2007.

[30] Jung, D., "Hierarchical Path Planning and Control of a Small Fixed-Wing UAV: Theory and Experimental Validation," Ph.D. Dissertation, School of Aerospace Engineering, Georgia Inst. of Technology, Atlanta, GA, Dec. 2007.

[31] Jung, D., Ratti, J., and Tsiotras, P., "Real-Time Implementation and Validation of a New Hierarchical Path Planning Scheme of UAVs via Hardware-in-the-Loop Simulation," *Journal of Intelligent and Robotic Systems*, Vol. 54, Nos. 1–3, 2009, pp. 163–181.
doi:10.1007/s10846-008-9255-0