Acta Cybernetica **27** (2026) 293–311.

Optimal Local Path Planner Over Receding Horizon Using Open Interval B-Spline*

Lucas Si Larbi[abc], Eric Lucet[ad], and Julien Alexandre dit Sandretto[be]

Abstract

A local path planning algorithm aims to provide a global, deterministic and safe solution for the dynamic navigation of wheeled robots with limited visibility. To this end, a novel approach based on open interval B-spline curves computed over a receding horizon is exposed in this paper. Real-time performances are ensured by using an interval branch and bound algorithm. The resulting path is smooth, obstacle-avoidant, and continuously connects local paths without requiring additional computation. Furthermore, this approach offers a guaranteed understanding of the solution's state. A large set of simulations, adapted for a wheeled differential robot on several scenarios, is finally carried out to assess parameters impact and performances.

Keywords: mobile robot, local path planner, interval B-Spline, non-linear problem, global optimization

1 Introduction

Autonomous off-road navigation of wheeled mobile robots requires rigorous control adapted to the robot's local environment. From classical methods such as sampling-based methods [21] or velocity-based methods [7]; to learning approaches such as genetic algorithms [13] or end-to-end machine learning methods [3], various approaches have been proposed in literature applied to autonomous navigation on uneven terrains. Main requirements in this context are: (i) The ability to consider multiple constraints such as geometrical constraints (*e.g.* curvature), or spatial constraints (*e.g.* avoid obstacles); (ii) The possibility to optimize the local path

*This work was carried out in the scope of REFLEX project, as part of the MOBILEX Challenge. This project received funding from the French Defense Innovation Agency (AID) and the French National Research Agency (ANR) and, in partnership with the French National Center for Space Studies (CNES) and the French Agency for Transport Innovation (AIT).

[a]Université Paris Saclay, CEA, List, F-91120 Palaiseau, France
[b]U2IS, ENSTA Paris, Institut Polytechnique de Paris, 91120 Palaiseau, France
[c]E-mail: lucas.silarbi@cea.fr, ORCID: 0009-0004-6045-9259
[d]E-mail: eric.lucet@cea.fr, ORCID: 0000-0002-9702-3473
[e]E-mail: julien.alexandre-dit-sandretto@ensta-paris.fr, ORCID: 0000-0002-6185-2480

DOI: 10.14232/actacyb.315468

with regard to the robot's local environment; (iii) The ability to guarantee an output solution, when one exists, or else the information of its non-existence; (iv) Finally, the ability to control the computation time to deal with real-time performances. In this context, and because of their rigorous properties, interval global optimization-based methods [9] appear to be a suitable choice [26]. Local planning usually comes within a more general architecture involving global planning and tracking control. Global planning aims to join the starting point with the end point, a goal, using the shortest path which often doesn't respect any particular constraint except avoiding *a priori* known obstacles. The role of tracking control is to generate motor commands in order to follow the solution generated by the motion planner. In many applications, a simpler architecture, only including a global planning level with a high refresh rate (A* [8], Rapidly exploring Random Tree [21]) and a tracking control level (Pure Pursuit [18], Model-based Predictive Controller [17]) is sufficient to obtain acceptable results. However, in unstructured environments, the gap between global planning and tracking control is important. Local planning level is therefore essential to generate a more complete solution over a finite horizon around the robot. This solution can take several forms: a path; a trajectory; a motion direction; or a velocity. Unlike the trajectory, a path is a succession of points that are not temporally linked. A robot following a path is then able to change its velocity, to stop and even reverse without having to regenerate it. This can be interesting in complex situations. This article is therefore focusing on path generation.

1.1 Related Work and Motivation

Some previous studies already considered B-Spline Curves (BSCs) in the generation of an optimized trajectory over a receding horizon [6, 19]. B-Spline Curves where chosen as the solution support due to their properties: a local modification; a definition of the entire curve only with several control points; a setting of the degree of continuity of the curve, and therefore the smoothness of the trajectory generated. The local planning problem has been formalized as a Numerical Constrained global Optimization Problem (NCOP). Sequential least squares quadratic programming [14] were used to find a solution of such problems. Two main issues were raised: the inability to guarantee a solution if one exists; and the inability to control computation time and therefore to ensure real-time performance. In this article, we focus on finding the right solver for our particular problem. There are several types of solver, in particular those based on artificial intelligence. However, these solvers do not address the previous issues. Also, interval methods applied to constrained global optimization allows one to find the global optimum and provide bounds on its value and location [9]. Moreover, interval branch and bound optimizers [2] can prematurely stop the optimization and provide a sub-optimal solution which respects constraints and therefore ensure real-time performance. Path planners using interval methods are already widely developed [1, 12]. The path planner proposed in this article addresses the real-time requirement for local motion planning. As previous studies have shown, reducing the search space is often essential

to limit the convergence time. In our approach, instead of using only the robot model as constraints to define the search space, we propose an adapted solution support. BSCs appear to be interesting for their properties. However, they are functions from real arithmetic, to be compatible with interval branch and bound, it is therefore required to used their interval extension.

1.2 Contribution

After an intuitive introduction to interval arithmetic and to B-spline curves in Section 2. We provide the first application of interval BSCs (introduced in Section 3) applied to path generation for mobile robots in Section 4. The formalization onto a NCOP and the optimization using an interval branch and bound are presented through three scenarios and a deep analysis based on 10K runs. Each figure of this article is rigorously and symbolically computed using Python and C++ programs.

2 Preliminaries

Interval arithmetic and B-splines need to be introduced to smoothly explain the new approach presented in this article.

2.1 Interval Arithmetic

The interval arithmetic is an extension of real arithmetic. Intervals (Definition 2.1) are considered instead of real numbers.

Definition 2.1. *Let* $\boldsymbol{x} = [\underline{x}, \overline{x}]$ *be an* ***interval*** *with its two bounds* $(\underline{x}, \overline{x}) \in \mathbb{R}^2$, $\underline{x} \leq \overline{x}$. $\boldsymbol{x}$ *contains all real numbers between* $\underline{x}$ *and* $\overline{x}$ *including them. The set of intervals is denoted* $\mathbb{IR}$. *If* $\underline{x} = \overline{x}$, $\boldsymbol{x}$ *is said to be degenerate and can be seen as a real number* $x = \underline{x} = \overline{x}$. *The width of* $\boldsymbol{x}$ *is* $w(\boldsymbol{x}) = \underline{x} - \overline{x}$. *The middle of* $\boldsymbol{x}$ *is denoted* $m(\boldsymbol{x}) = \frac{\underline{x}+\overline{x}}{2}$. *Finally, a box* $\boldsymbol{X} = (\boldsymbol{x_0}, ..., \boldsymbol{x_n})$ *is a vector of intervals.*

The interval arithmetic was first proposed in [20]. It was then fully extended to deal with real arithmetic issues and limitations such as the division by zero or the definition and representation of infinite numbers. Moreover, the interval arithmetic appears to be a strong solution to perform robust and guaranteed computer calculation [29]. In particular, it offers a solution to the classic rounding error occurring with numerical representation of real numbers in a finite number of bits: the outward rounding [11, 22]. Finally, like other arithmetics, it provides a wide range of operations and tools making it interesting for many applications [12]. For our use-case, we use both interval objects and tools such as respectively interval B-spline curve [25] and interval branch and bound optimizer [2].

Notation 1. *For clarity and to avoid mismatch between real numbers, intervals and boxes, the following notations are used in this article:* x *is a real number;* $\boldsymbol{x}$ *is an interval;* X *is a vector; and* $\boldsymbol{X}$ *is a box.*

2.1.1 Operations

The interval arithmetic includes interval extension of real operations which are subject to a theorem (Theorem 2.1).

Theorem 2.1. *A real operation* $\diamond \in \{+, -, \times, \div, ...\}$ *can be extended to the interval arithmetic if* $\boldsymbol{x} \diamond \boldsymbol{y}$ *is an interval containing* $\{x \diamond y \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\}$.

$$\boldsymbol{x} \diamond \boldsymbol{y} = [\{x \diamond y \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\}] \tag{1}$$

For example, some operations are defined as follows:

- Addition:
$$\boldsymbol{x} + \boldsymbol{y} = [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \tag{2}$$
- Subtraction:
$$\boldsymbol{x} - \boldsymbol{y} = [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \tag{3}$$
- Multiplication:
$$With\ \mathcal{D} = \{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\},\ \boldsymbol{x}\boldsymbol{y} = [min(\mathcal{D}), max(\mathcal{D})] \tag{4}$$

As interval arithmetic deals with sets, operations such as the intersection $\cap$, the union $\cup$ and the inclusion $\subset$ are also available.

2.1.2 Dependency

The interval arithmetic suffers from an important weakness called the dependency problem. This problem appears because interval operations are too conservative.

Example 2.1. Let $\boldsymbol{x}$ be a non degenerate interval. Then, $\boldsymbol{x} - \boldsymbol{x} = [\underline{x} - \overline{x}, \overline{x} - \underline{x}]$. Therefore, $\boldsymbol{x} - \boldsymbol{x} \neq [0, 0]$.

Although $0 \in \boldsymbol{x} - \boldsymbol{x}$, $\boldsymbol{x} - \boldsymbol{x}$ is not equal to $[0, 0]$ (Example 2.1) because, according to the Theorem 2.1, the two operation members are considered independent. In other words, the operation $\boldsymbol{x} - \boldsymbol{x}$ is equivalent to $[\{a - b \mid a \in \boldsymbol{x}, b \in \boldsymbol{x}\}]$. More generally, the dependency problem appears when the number of occurrences of a variable in an equation is greater than one [20]. The dependency problem is well known and other arithmetics, such as affine arithmetic [5] or generalized interval arithmetic [9], were proposed to reduce its impact [15]. In Section 3, we explain how the definition of interval B-spline curve gets around the dependency problem.

2.1.3 Interval Function

As well as real numbers, the interval arithmetic provides an interval extension of functions.

Definition 2.2. *Considering* $f : \mathbb{R}^n \to \mathbb{R}^m$*, an extension or inclusion function is the* ***interval function*** $\boldsymbol{f} : \mathbb{IR}^n \to \mathbb{IR}^m$ *if:*

$$\forall \boldsymbol{x} \in \mathbb{IR}^n, f(\boldsymbol{x}) \subset \boldsymbol{f}(\boldsymbol{x}) \tag{5}$$

In this article, we consider the natural interval extension, which consists in substituting real operators with their interval equivalents presented in Section 2.1.1.

Remark 2.1. Because of the dependency problem, the syntax of the equation has an important impact on the result interval width. It is therefore interesting to use symbolic computation in order to reduce the dependency problem.

2.2 Interval Branch and Bound

The interval arithmetic opens the way to an interesting class of optimizers called Interval Branch and Bound (IB&B) [2]. These optimizers use a succession of bisections and contractions on boxes to optimize NCOPs with the guarantee to provide bounds which enclose the global optimum of the cost function [9] (Remark 2.2). A bisection consists in separating a box into two child boxes. A contraction is an operation which, using the analytical expression of constraints, reduces the widths of the NCOPs variables. Such optimization can be stopped prematurely (Remark 2.3). Although sub-optimal, the resulting solution satisfies all the constraints. This capability enables to deal with real-time performance. Moreover, both parts of a box resulting from a bisection can be processed independently. IB&B are therefore parallelizable.

Remark 2.2. When the optimization process ends within the allocated time, IB&B guarantees the following returns:

- Bounds of the cost function global optimum and corresponding optimized variables, if they exist.
- An empty set, if no valid solution exists.

Remark 2.3. When the optimization is prematurely stopped, IB&B can deliver two different results:

- If no point validating constraints was found during the allocated time: the algorithm returns infinite bounds for the global optimum of the cost function.
- If at least one point validating constraints is found during the allocated time: the IB&B returns sub-optimal bounds of the cost function global optimum and corresponding optimized variables.

2.3 Bézier and B-spline curves

Bézier and B-spline Curves (BCs & BSCs), their generalization, are part of the polynomial curves family. They are widely used by the computer aided design community; due to their numerous properties and because they offer a powerful

basis for shapes representations. In this section, we present some interesting properties of BCs and BSCs for path construction and optimization application. More BCs and BSCs properties, rigorous demonstrations and references can be found in books [23, 24]. As well as every polynomial curves, BCs and BSCs can be represented by either an implicit equation or a parametric function. Unlike implicit equation, the parametric function form separately describes each variable with a function depending on a joint parameter[3]. BCs and BSCs can be represented by both implicit equation and parametric function forms. In this article, we only consider the parametric function form because it simplifies programming and makes curve handling more intuitive and flexible. BCs must be introduced (Definition 2.3) in order to better understand BSCs properties and advantages.

Definition 2.3. *A **Bézier curve** is a polynomial given by:*

$$C(u) = \sum_{i=0}^{n} B_{i,n}(u) P_i, \ u \in [0,1] \tag{6}$$

with:

- *the Berstein Polynomial Basis Functions (BPBFs):*

$$B_{i,n}(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i} \tag{7}$$

- *the control points:*

$$P_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{8}$$

BCs are defined by a finite number of control points which allow, when their coordinates are modified, to change the shape of the curve. Moreover, as BCs are constructed from Bernstein polynomials, curves' degree of continuity is controlled by the degree of polynomials. The number of control points thus directly influences the curve degree. Figure 1 shows an example of a three-degree BC (four control points), starting with its BPBFs, influenced by the attraction of control points, on lateral planes (xu, yu). The sum of the BPBFs over x and y can be respectively identified on both planes by a blue and a red dotted curve. The projection of these two sums onto the xy plane (visible at the back in green) represents the final BC. BPBF computation details are provided in Table 1.

BCs suffer from an important drawback: the degree of the curve is directly related to the number of control points. Thus, the degree of the curve quickly increases when a more precise control is required. Furthermore, by definition, BCs consist of a single segment: each point of the curve is influenced by every control points. Consequently, when a control point is moved, the entire curve is modified.

[3] One parameter per dimension. For example two parameters for a surface.

Table 1: Bézier curve construction example (related to Figure 1)

$B_{i,n}(u) * P_i$	$x_0 = 0.5$ $y_0 = 0.2$	$x_1 = 1.5$ $y_1 = 1.0$	$x_2 = 2.3$ $y_2 = 0.5$	$x_3 = 2.8$ $y_3 = 1.0$
$B_{1,n}(u) = -(u-1)^3$	(0) $-0.5(u-1)^3$ (0) $-0.2(u-1)^3$	-	-	-
$B_{2,n}(u) = 3u(u-1)^2$	-	(1) $4.5u(u-1)^2$ (1) $3u(u-1)^2$	-	-
$B_{3,n}(u) = -3u^2(u-1)$	-	-	(2) $-6.9u^2(u-1)$ (2) $-1.5u^2(u-1)$	-
$B_{4,n}(u) = u^3$	-	-	-	(3) $2.8u^3$ (3) u^3
$C(u) = \sum_{i=0}^{n} B_{i,n}(u)P_i$	$(\sum_x)$ $\frac{-(u^3+6u^2-30u-5)}{10}$ $(\sum_y)$ $\frac{(23u^3-39u^2+24u+2)}{10}$			

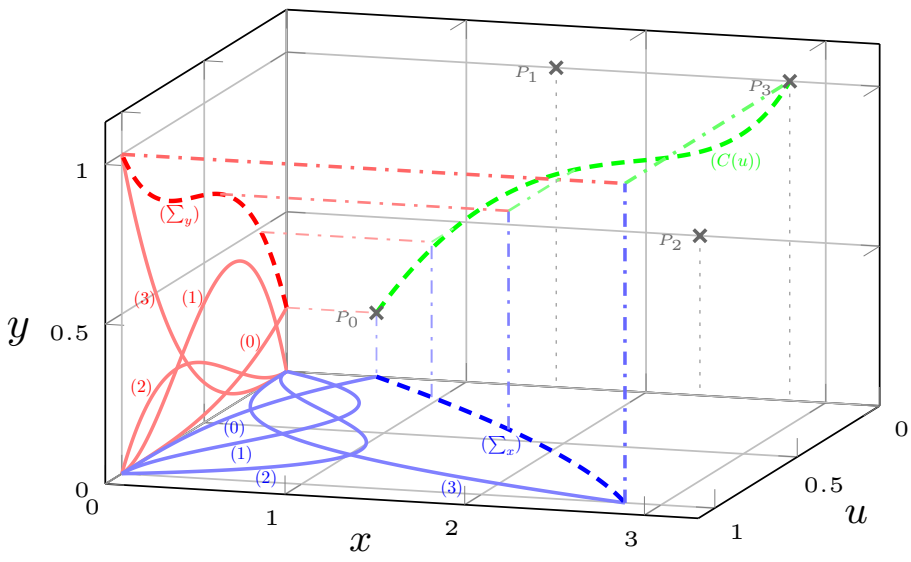


Figure 1: Bézier curve construction example (caption in Table 1)

BSCs were introduced in order to address these issues. Instead of BPBFs, they are constructed from the sum of B-Spline Basis Functions (BSBFs) computed by recurrence (Definition 2.4).

Definition 2.4. *A **B-spline curve** is a connection of polynomials given by:*

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i, \ u \in [u_p, u_{n+1}] \tag{9}$$

with:

- *the B-spline basis functions:*

$$N_{i,0}(u) = \begin{cases} 1 & if \quad u_i \leq u \leq u_{i+1} \\ 0 & otherwise \end{cases} \tag{10}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{11}$$

- *the knot vector:*

$$\mathcal{U} = \{u_0, u_1, ..., u_{n+p+1}\}, \ u_i \leq u_{i+1} \tag{12}$$

- *the control points:*

$$P_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{13}$$

As the parameter u moves in the sequence $\mathcal{U}$, a BSC is made up of connected segments, hence the name: nodes, for the terms of $\mathcal{U}$. A pth-degree BSC controlled by $n + 1$ control points has $n + 1 - p$ segments. Unlike BCs, the degree p of BSCs is not linked to the number of control points $(n + 1)$. Thus, when a control point is moved, only $p + 1$ segments are modified. It allows local modification and drastically reduces the amount of computation. Moreover, the continuity degree and therefore the smoothness of the curve are fully controllable. Figure 2 shows four BSCs. The three solid lines represent a three, five and eight-degree BSCs. Thus, the degree impact on curve deployment is observable. Moreover, in order to show the local modification property of BSCs, a control point has been moved (P_4 to P_4'), generating the three-degree dashed curve. Finally, each color on visible BSCs represents a segment.

Remark 2.4. BCs and BSCs can be seen as an approximation of their control points. An important and well-known property follows: curves are included in the envelope built from their control points.

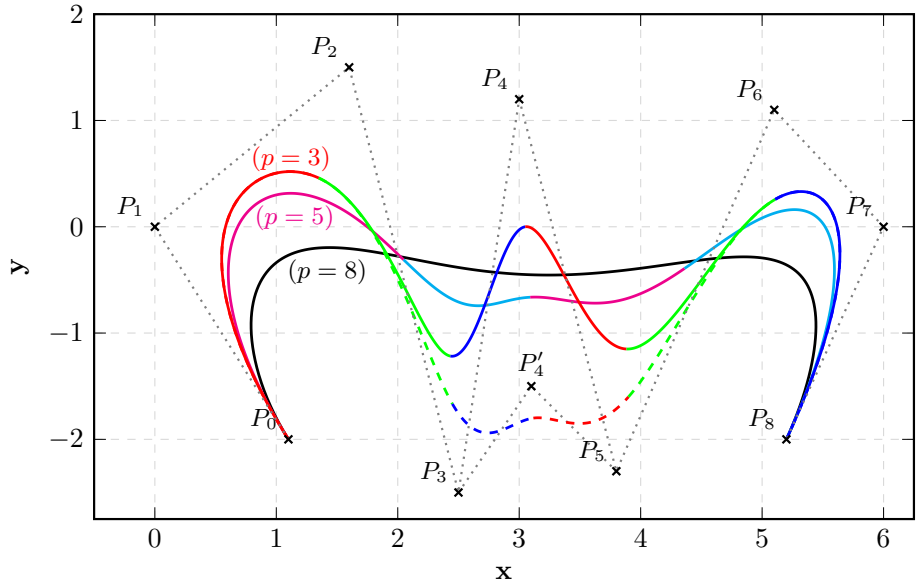


Figure 2: View of three BSC of different degrees (3, 5, 8). View of the impact of control points positioning on curve deployment (dashed curve).

2.4 Clamped, Semi-Open and Open B-spline

The knot vector $\mathcal{U}$ has an important impact on the spatial expansion and the continuity degree of the BSC. The curve switches from one segment to another. Each segment corresponds to the parameter displacement between two nodes. Thus, the knot vector directly influences the position and the continuity degree at segment connections. When a node is repeated multiple times in $\mathcal{U}$, this node multiplicity increases. A node of multiplicity m on a pth-degree BSC means that polynomials to the left and right of the node are of continuity equal to $p-m$. An higher multiplicity therefore implies a loss of continuity. In most applications, the multiplicity of the first and last nodes are fixed to $p+1$. The rest of the knot vector is chosen to be uniform (each node has a multiplicity of one and is evenly distributed in $\mathcal{U}$). This generates well-known clamped BSCs (Figure 2). Clamped means that the curve touches the first and last control points. Thus, the BSC end positions are better controlled. However, when it comes to connecting several clamped BSCs, additional operations are required to ensure continuity (pth-degree derivatives, curvatures and curvature evolution must be equal on the left and right of the connection).

Remark 2.5. A pth-degree clamped BSC controlled by $p+1$ control points is a BC. Moreover, its knot vector has only two nodes (0 and 1) of multiplicity $p+1$. *E.g.*: The 8th-degree clamped BSC visible in the Figure 2 (black curve) is a BC.

When it comes to making a large number of connections between BSCs, it is therefore better to keep the knot vector fully uniform and thus generate open BSCs.

Indeed, as the open BSC has a uniform knot vector, the continuous connection is therefore automatic and does not require additional computation. In fact, in order to connect two pth-degree open BSCs, p consecutive control points need to be equal between the two BSCs. For example, the Figure 3(2) shows two connected open BSCs of degree three. To do so, the first, second and third control points of the red BSC were chosen equal to the second, third and fourth control points of the blue one. In this case, the connection takes place at the end of the first segment of the blue BSC. In general, by definition, connections are necessarily made at the junction between segments. There's also an hybrid version where only the first node of $\mathcal{U}$ has a multiplicity of $p+1$, resulting in a semi-open BSC. Three-degree clamped, semi-open and open BSCs, generated with the same control polygon (list of control points) are visible in the Figure 3(1).

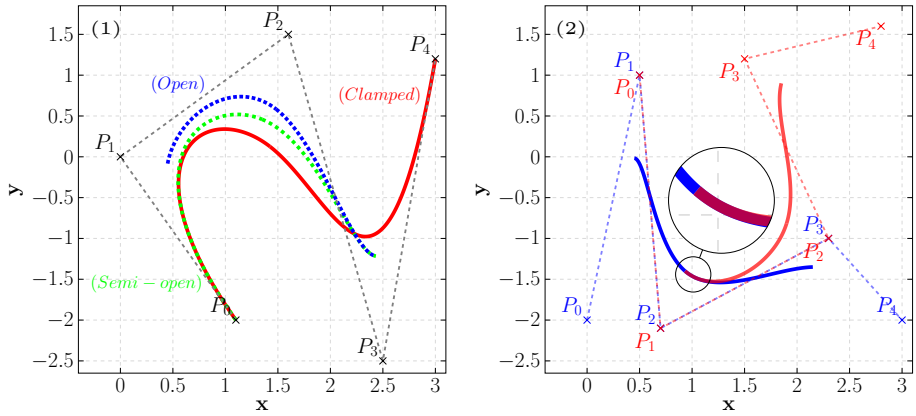


Figure 3: Clamped, semi-open and open B-splines (left). Connection of two open B-spline curves(right).

3 Interval B-spline

Interval B-spline Curves (IBSCs) were introduced in [25] (Definition 3.1) and mainly used for data approximation in various contexts: reverse modeling in computer aided design [28]; curve reconstruction [16]; or more recently for hyperspectral imaging [4].

Definition 3.1. *An **interval B-spline curve** is a B-spline curve in which the control points are boxes (Figure 4):*

$$\boldsymbol{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\boldsymbol{P_i} \tag{14}$$

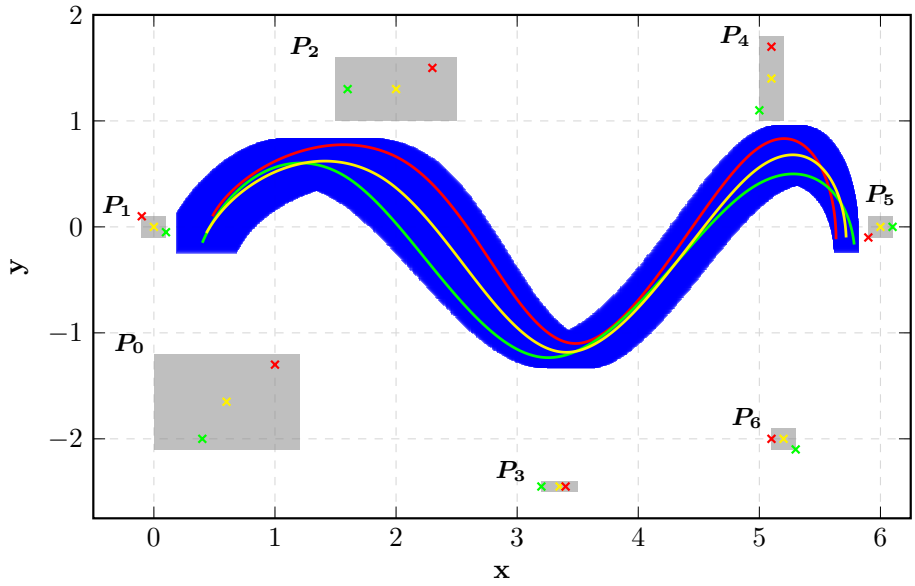


Figure 4: IBSC with three included BSCs including it's midpoint

Property 1. *The coordinates of an IBSC are obtained by algebraic operations defined in the interval arithmetic (Section 2.1.1). Consequently, considering an IBSC* $\boldsymbol{C_Q}$ *build with control points included in control points of another IBSC* $\boldsymbol{C_P}$*.* $\boldsymbol{C_Q}$ *is thus included in* $\boldsymbol{C_P}$ *and is an IBSC:*

$$\forall \boldsymbol{Q_i} \subset \boldsymbol{P_i} \implies \boldsymbol{C_Q}(u) = \sum_{i=0}^{n} N_{i,p}(u)\boldsymbol{Q_i} \subset \boldsymbol{C_P}(u) = \sum_{i=0}^{n} N_{i,p}(u)\boldsymbol{P_i} \tag{15}$$

Remark 3.1. A curve included in an IBSC is not necessarily an IBSC.

The definition 3.1 is interesting as it avoids the dependency problem. Indeed, as the parameter u is a real number, the equation of the curve can be symbolically simplified. In its simplest form, each variable, corresponding to the control points coordinates, appears only once in the equation. The size of the box resulting from the curve evaluation is then minimal [20]. The dependency problem is overcome.

4 IBSC-based Path Planner

Our local planning approach consists in optimizing successive open IBSC using an IB&B to generate paths over a receding horizon. Each optimized IBSC is automatically connected with the previous one using the method presented in Section 2.4. Our application uses circular obstacles to simplify constraints. Distances between curves and obstacles are therefore simpler to compute. We use the Ibex library [10]

and its plugin IbexOpt, which respectively implement the interval arithmetic and an IB&B in C++. Symbolic calculations are made with SymEngine [27] in C++.

4.1 NCOP for path planner

The objective is to connect the starting point S to the end point E with a finite number of optimizations n_{opt}. The environment contains n_{obs} circular obstacles of center Ωo_j and radius ro_j. During the k^{th} optimization, the robot has a known perception distance resulting in a circular planning horizon H_k, centered on the robot position Ωh_k, and of radius rh_k. Only visible obstacles: V_{obs} (included in H_k), are considered for the k^{th} optimization.

Notation 2. *The euclidean distance between A and B is denoted* $d(A, B)$.

The k^{th} NCOP is defined as:
Variables:

$$\boldsymbol{P_{i,k}} = \left| \begin{array}{l} S,\ if \left\{ \begin{array}{l} i \in \{0, ..., p-1\} \\ k = 0 \end{array} \right. \\ \boldsymbol{P_{i+s,k-1}},\ with\ s \in \{1, ..., n-p\},\ if \left\{ \begin{array}{l} i \in \{0, ..., p-1\} \\ k \in \{1, ..., n_{opt}-1\} \end{array} \right. \\ \left[\begin{array}{l} [-\infty, +\infty] \\ [-\infty, +\infty] \end{array} \right] \ otherwise \end{array} \right. \tag{16}$$

Cost function:

$$\boldsymbol{z_k} = d^2(\boldsymbol{C}(u_{n+1}), E) + \sum_{i=0}^{n-1} d^2(\boldsymbol{P_{i,k}}, \boldsymbol{P_{i+1,k}}) \tag{17}$$

Constraints:

$$\Gamma_k = \left\{ \begin{array}{l} \forall (\Omega o_j, ro_j) \in V_{obs},\ \forall u,\ d^2(\boldsymbol{C}(u), \Omega o_j) \subset \left[ro_j^2, +\infty\right] \\ d^2(\Omega h_k, \boldsymbol{C}(u_{n+1})) \subset \left[rh_k^2 - \varepsilon, rh_k^2\right],\ \varepsilon > 0 \end{array} \right. \tag{18}$$

Remark 4.1. All distances are squared to avoid square roots in the symbolic form of functions, reducing the number of operations during the contraction.

Variables of the k^{th} optimization correspond to the control points of the k^{th} IBSC (Eq. (16)). By definition, the first point of the curve is affected by p control points. During initialization, setting the latter equal to the starting point S ensures that the IBSC starts at S. For other optimizations, to ensure a continuous connection, the p first control points must correspond to p successive control points of the previous optimized IBSC. As discussed in Section 2.4, the connection between two curves necessarily takes place at the junction between segments. The variable s is used to specify after which segment of the k^{th} curve the connection with the $k+1^{th}$

one takes place. For example, if $s = 1$, the connection takes place at the end of the first segment of the k^{th} curve. Secondly, we want to minimize: (i) The distance between the last point of the curve and the end point E, to gradually connect it to E; (ii) The distance between successive control points to reduce the envelop (control polygon) area and therefore the spatial expansion of the curve. The cost function is thus built from these two elements (Eq. (17)). Finally, all curve coordinates must be outside obstacles and the last point of the curve must be close to the planning horizon. These two conditions generate the NCOP constraints (Eq. (18)). In order to exactly reach the end point, when E is inside the planning horizon, the planning horizon is set equal to the distance between the robot and E. The end of the curve thus does not exceed E. The planner succeeds when the last point of the curve is inside an area around E.

4.2 Path classification

When an IB&B provides a solution, it guarantees that constraints are validated. In the same way, it is possible to detect when optimization has failed (Remarks 2.2 and 2.3). Strategies in case of failure can then be considered. In our approach, as an optimization takes place in a finite time ($t < timeout$), three scenarios can be considered, leading to four path planner states:

- The solution is valid: it is stored and the following optimization is triggered. If the end point is reached, the path planner succeeded ($state = \boldsymbol{valid}$). If the generated path returns to the same location several times, the path planner is blocked in a concavity formed by obstacles. In practice, to detect this phenomenon, distances between the center of the new horizon to the center of all the previous horizons are observed. If the next planning horizon is close to several previous horizons (several of these distances are smaller than a threshold): the path planner is considered blocked ($state = \boldsymbol{blocked}$).
- There is no solution (empty set): optimizations are stopped ($state = \boldsymbol{fail}$).
- There is no solution in $t < timeout$ ($w(\boldsymbol{z_k}) = +\infty$): a local start point is defined as $S_k = \Omega h_k$. A new optimization is initiated with more time allocated. If it succeeds, the next optimization is triggered ($state = \boldsymbol{fail\ recovered}$). The fail recovered state should be seen as an attempt to take into account rare situations occurring in real-life conditions. Other solutions could be considered, such a request for assistance involving human intervention. Our approach disrupts the smoothness of the path, introducing a continuity break. This recovery approach reduces the scope of suitable robots to differential and holonomic. If it fails again or if the number of fail recovered exceeds a threshold value: optimizations are stopped ($state = \boldsymbol{fail}$).

4.3 Experiments

In this section we analyze this approach in two experiments. Firstly, three general scenarios are presented to analyze the behavior of the local planner. Secondly, an

analysis of a large number of simulations involving random obstacles is proposed to study the states distribution.

4.3.1 Three scenarios analysis

Results of three scenarios are displayed in Figure 5 to examine how the time allocated to optimization (timeout) and the planning horizon (rh_k) affect the path planner behavior. Values of other parameters are set as follows:

- $p = 3$: A 3-degree IBSC is C^2, which ensures a continuous curvature and a smooth path.
- $n = 4$: The number of control points affects directly the problem dimension (e.g. $n = 4$: problem of dimension $2(n + 1) = 10$). Furthermore, the number of control points impacts the number of segments. The IBSC must have a minimum of 2 segments to use the principle generation over receding horizon. A 3-degree IBSC built from 5 control points ($n = 4$) has 2 segments and therefore limits the problem dimension.
- $s = 1$: A 3-degree IBSC build from 5 control points has 2 segments, s is necessarily equal to 1. The connection between the k^{th} and the $k+1^{th}$ IBSC therefore takes place after the first segment of the k^{th} curve.
- u is discretized 25 times per segment. This gives an average offset of $20cm$ between each curve evaluation (assumption of a straight curve over a $10m$ planning horizon).

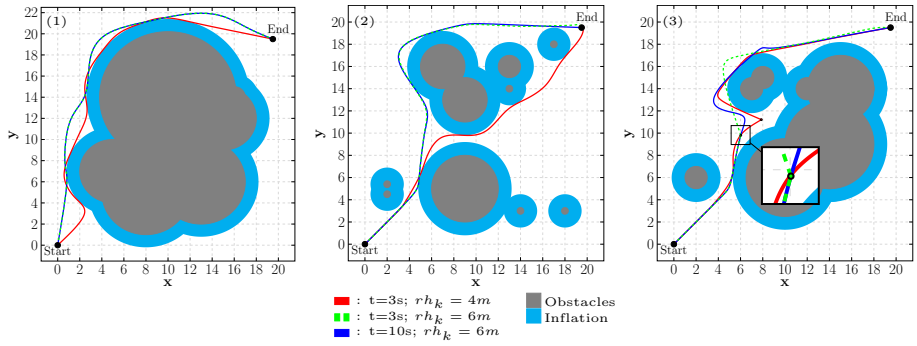


Figure 5: Timeout (t) and planning horizon (rh_k) impacts on the local planner behavior

Each scenario has a specific environment: (1) Environment composed of large obstacles forming a single cluster. (2) Cluttered environment with small obstacles. (3) Cluttered environment where obstacles form a dead end. Three parameter sets were tested in different environments to observe the timeout and planning horizon impact on the path planner behavior. The shorter the planning horizon, the closer

the path is to obstacles. An IBSC with a small number of control points generated over a large horizon means that a single control point causes deformation over a large area, resulting in some quite bulging areas close to obstacles. Furthermore, a shorter horizon allows the planner to pass through less accessible areas. Scenario (3) shows two ***fail recovered*** states. In both cases, when entering the dead end, the optimizer did not obtain a solution within the allocated time. A breaking point was set up (equivalent to a new starting point), and a new optimization was started with more time allocated. The blue curve does not contain a break. Indeed, due to a larger timeout, the optimizer found a more complex solution. Scenarios (1) and (2) show that the timeout has little impact in environments without dead ends.

4.3.2 Simulated states distribution

A large number of simulations have been carried out with random obstacles and different parameter sets. An environment, similar to those shown in Figure 5, is randomly generated for each simulation. Results are shown as four matrices in Figure 6. One hundred simulations were carried out for each pair of parameters. For a given pair on a given matrix, the resulting cell represents the state distribution. For example, with the pair $(t = 10, rh_k = 1)$, we obtained 56 % valid; 0 % fail; 0 % fail recovered; and 44 % blocked. This study shows that the choice of parameters

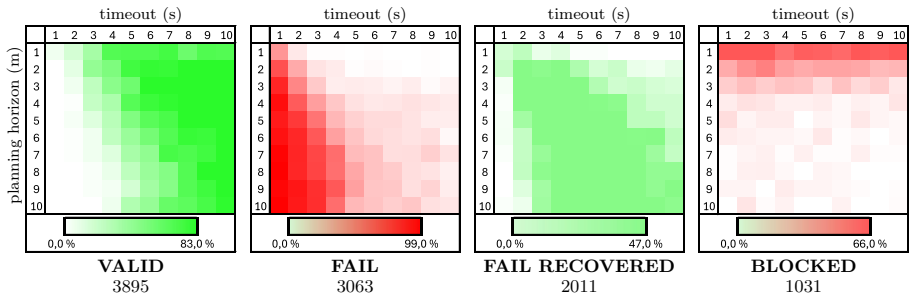


Figure 6: 10K simulations analysis

is important to obtain reliable results. In fact, for a timeout greater than $3s$ and a planning horizon greater than $3m$, more than 77 % of simulations reach the target while avoiding obstacles. In cases of low timeouts, IB&B doesn't have time to obtain a solution, thus the number of fails is important. Furthermore, it is logical to observe that scenarios with a small planning horizon tend to get blocked. As part of a complete mobile robotics application, a global planner would provide intermediate waypoints, helping the local planner to avoid dead ends. A larger planning horizon implies more calculation, as more obstacles are taken into account. Thus, for the same timeout, the number of valid solutions decreases when the horizon increases. In this case, as the fail recovery procedure increases the timeout, many fails are saved and the target is still reached.

4.4 Discussion

A limitation comes from the discretization resolution of the parameter u in the IBSC definition. The gap between two IBSC evaluations makes obstacle avoidance constraints less rigorous. In theory, the obstacle avoidance constraint (Eq. (18)) guarantees that for all parameters u, the IBSC is outside obstacles. However, the coordinate of a point resulting from the curve evaluation for a parameter somewhere between two successive values of u is not necessarily outside obstacles. The lower the discretization resolution, the greater the risk of slightly entering an obstacle. Moreover, the amount of computations increases proportionally with the discretization resolution of u. This resolution must therefore be chosen according to the size of the obstacles. An improvement would be to no longer consider u as a real but as an interval. In this case, obstacle avoidance would be ensured. However, the problem of dependency would no longer be overcome. A new definition of IBSC would be therefore required. Another limitation comes from the circular obstacle assumption. Ideally, all types of obstacle should be considered. One improvement would be to apply a cost function directly on a cost map and use a binary map for constraints. In this way, all obstacles would be taken into account with the resolution of the binary map.

5 Conclusion

Autonomous off-road navigation of wheeled robots requires rigorous control and planning approaches. The local planning level is essential to fill the gap between the global planning and the tracking control levels. Interval methods are interesting in this context for their reliability and provided guarantees. We propose a new interval-based local path planner. The local path is obtained by successive optimization of an open interval B-spline curve over receding horizon using a branch & bound optimizer. We thus propose the first application of open interval B-spline curves for path generation in robotics. After introducing B-splines and interval arithmetic, interval B-splines were intuitively presented. The formalization, a demonstration in three scenarios, and an in-depth study involving 10K simulations have been proposed. This new approach has proved its efficiency in complex situations involving dead-ends and cluttered environments. Furthermore, our method provides a guaranteed verdict on the solution state obtained after optimization (valid, fail, fail recovered, or blocked). This technique is general and modular. Various constraints and optimization criteria can be applied to customize the resulting path. The requirements of navigation on uneven terrains are then respected.

References

[1] Alexandre Dit Sandretto, J., Chapoutot, A., and Mullier, O. Formal verification of robotic behaviors in presence of bounded uncertainties. In *Proceedings*

of the First IEEE International Conference on Robotic Computing, pages 81–88. IEEE Computer Society, 2017. DOI: 10.1109/IRC.2017.17.

[2] Araya, I. and Reyes, V. Interval branch-and-bound algorithms for optimization and constraint satisfaction: A survey and prospects. *Journal of Global Optimization*, 65(4):837–866, 2016. DOI: 10.1007/s10898-015-0390-4.

[3] Benatti, S., Young, A., Elmquist, A., Taves, J., Tasora, A., Serban, R., and Negrut, D. End-to-end learning for off-road terrain navigation using the Chrono open-source simulation platform. *Multibody System Dynamics*, 54(4):399–414, 2022. DOI: 10.1007/s11044-022-09816-1.

[4] Boukezzoula, R., Coquin, D., and Jacq, K. A possibilistic regression based on gradual interval B-Splines: Application for hyperspectral imaging lake sediments. *Information Sciences*, 510:183–204, 2020. DOI: 10.1016/j.ins.2019.09.031.

[5] de Figueiredo, L. H. and Stolfi, J. Affine arithmetic: Concepts and applications. *Numerical Algorithms*, 37(1):147–158, 2004. DOI: 10.1023/B:NUMA.0000049462.70970.b6.

[6] Defoort, M., Palos, J., Kokosy, A., Floquet, T., and Perruquetti, W. Performance–based reactive navigation for nonholonomic mobile robots. *Robotica*, 27(2):281–290, 2009. DOI: 10.1017/S0263574708004700.

[7] Douthwaite, J. A., Zhao, S., and Mihaylova, L. S. A comparative study of velocity obstacle approaches for multi-agent systems. In *Proceedings of the UKACC 12th International Conference on Control*, pages 289–294, 2018. DOI: 10.1109/CONTROL.2018.8516848.

[8] Foead, D., Ghifari, A., Kusuma, M. B., Hanafiah, N., and Gunawan, E. A systematic literature review of A* pathfinding. *Procedia Computer Science*, 179:507–514, 2021. DOI: 10.1016/j.procs.2021.01.034.

[9] Hansen, E. and Walster, G. W. *Global Optimization Using Interval Analysis: Revised And Expanded*. CRC Press, 2004. DOI: 10.1201/9780203026922.

[10] Ibex Team. Ibex documentation. URL: https://ibex-team.github.io/ibex-lib/intro.html#.

[11] IEEE. Standard for binary floating-point arithmetic. *ANSI/IEEE Std 754-1985*, pages 1–20, 1985. DOI: 10.1109/IEEESTD.1985.82928.

[12] Jaulin, L., Kieffer, M., Didrit, O., and Walter, E. *Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer London, 2001. DOI: 10.1007/978-1-4471-0249-6.

[13] Katoch, S., Chauhan, S. S., and Kumar, V. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126, 2021. DOI: 10.1007/s11042-020-10139-6.

[14] Kraft, D. *A Software Package for Sequential Quadratic Programming*. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, 1988. url: https://books.google.fr/books?id=4rKaGwAACAAJ.

[15] Krämer, W. Generalized intervals and the dependency problem. *Proceedings in Applied Mathematics and Mechanics*, 6(1):683–684, 2006. DOI: 10.1002/pamm.200610322.

[16] Lin, H., Chen, W., and Wang, G. Curve reconstruction based on an interval B-Spline curve. *The Visual Computer*, 21:418–427, 2005. DOI: 10.1007/s00371-005-0304-4.

[17] Lucet, E., Micaelli, A., and Russotto, F.-X. Accurate autonomous navigation strategy dedicated to the storage of buses in a bus center. *Robotics and Autonomous Systems*, 136:103706, 2021. DOI: 10.1016/j.robot.2020.103706.

[18] Macenski, S., Singh, S., Martín, F., and Ginés, J. Regulated pure pursuit for robot path tracking. *Autonomous Robots*, 47(6):685–694, 2023. DOI: 10.1007/s10514-023-10097-6.

[19] Mendes Filho, J. M., Lucet, E., and Filliat, D. Real-time distributed receding horizon motion planning and control for mobile multi-robot dynamic systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 657–663, 2017. DOI: 10.1109/ICRA.2017.7989081.

[20] Moore, R. E. *Interval Analysis*. Prentice-Hall series in automatic computation. Prentice-Hall, 1966. ISSN: 2577-9435.

[21] Orthey, A., Chamzas, C., and Kavraki, L. E. Sampling-based motion planning: A comparative review. *Annual Review of Control, Robotics, and Autonomous Systems*, 7:285–310, 2023. DOI: 10.1146/annurev-control-061623-094742.

[22] Palmer, J. F. and Morse, S. P. *The 8087 Primer*. Wiley, 1984. ISBN: 9780471875697.

[23] Piegl, L. and Tiller, W. *The NURBS Book*. Monographs in Visual Communication. Springer Berlin Heidelberg, 1996. DOI: 10.1007/978-3-642-59223-2.

[24] Risler, J. J. *Mathematical Methods for CAD*. Recherches en mathématiques appliquées. Cambridge University Press, 1992. ISBN: 9780521436915.

[25] Shen, G. and Patrikalakis, N. M. Numerical and geometrical properties of interval B-Splines. *International Journal of Shape Modeling*, 04(01n02):35–62, 1998. DOI: 10.1142/S0218654398000040.

[26] Si Larbi, L., Lucet, E., and Alexandre dit Sandretto, J. Overview of motion planning techniques and their suitability for an off-road navigation use-case. In *Proceedings of the 18th International Conference on Control, Automation,*

Robotics and Vision, pages 1054–1061, 2024. DOI: 10.1109/ICARCV63323.2024.10821670.

[27] SymEngine Team. SymEngine documentation. URL: https://symengine.org/.

[28] Tuohy, S. T., Maekawa, T., Shen, G., and Patrikalakis, N. M. Approximation of measured data with interval B-Splines. *Computer-Aided Design*, 29(11):791–799, 1997. DOI: 10.1016/S0010-4485(97)00025-0.

[29] Van den Berg, J. B. and Queirolo, E. Rigorous validation of a Hopf bifurcation in the Kuramoto–Sivashinsky PDE. *Communications in Nonlinear Science and Numerical Simulation*, 108:106133, 2022. DOI: 10.1016/j.cnsns.2021.106133.