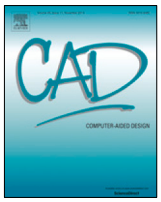
Computer-Aided Design 154 (2023) 103421

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad



Application Paper

A New Framework for Joint Trajectory Planning Based on Time-Parameterized B-Splines

Marco Riboli [a,*], Fabio Corradini [a,b], Marco Silvestri [a,b], Alessandra Aimi [c]

[a] Department of Engineering and Architecture, University of Parma, Italy
[b] Department of Innovative Technologies, University of Applied Sciences of Southern Switzerland, SUPSI, Lugano, Switzerland
[c] Department of Mathematical, Physical and Computer Science, University of Parma, Italy

ARTICLE INFO

Article history:
Received 7 November 2021
Received in revised form 4 June 2022
Accepted 19 September 2022

Keywords:
Trajectory planning and control
Cam mechanism
Univariate B-spline
Time-parameterized curve

ABSTRACT

This paper discusses a new, highly effective trajectory planning paradigm for robotics and manufacturing systems. The proposed approach offers a more flexible solution to a problem classically solved in closed-form, defining a generalization of the existing methods by providing a common data structure based on time-parameterized curves to represent exactly or approximately the joint trajectories. In the framework, the most generic approach to constrain or optimize a trajectory is to set up an iterative loop and act on the parameters through computer-aided design concepts, such as shape modeling, symbolic calculation, derivation and integration. The iso-parametric representation allows to define these procedures as macro-instructions to develop scalable and reusable solutions. A benchmark is proposed to confirm the advantages of the new approach over an established solution. Finally, an experimental case study of a single axis motion control is provided to illustrate and clarify the method.

1. Introduction

Motion planning is a crucial issue in the fields of robotics and manufacturing systems. Industrial processes require ever higher operating speeds for throughput maximization, which in turn depends upon refined motion design to preserve accuracy and repeatability. The complexity of the problem lies in the need to satisfy a combination of constraints linked together by non-linear relationships. In the Cartesian space, it is necessary avoiding obstacles, following a desired tool-path or adopt a controlled tool speed. At the same time, in the joint space the adopted actuators and, more generally, the system mechanical performances impose their own kinematic and dynamic limits. This difficulty leads to the definition of two main ways to plan trajectories: one that interpolates the end-effector path in the Cartesian space starting from a series of via-points and then returns the trajectories to the joints (thus solving the inverse kinematics every time step), and the other where the interpolation takes place in the joint space. The choice of strategy to be adopted depends solely on the specific application and the constraints to be imposed on motion. Even if a trajectory planning in Cartesian space appears more intuitive and allows a direct visualization of the end-effector path, this often fails due to kinematic singularity problems [1] and can return non-smooth timing laws of the axis. For this reason, it is sometimes preferable to plan the trajectory in the joint space by interpolating (or passing close to) a series of Cartesian points, but not ensuring the path between them due to the non-linearities introduced by the direct kinematic operation [2,3]. From the implementation point of view, the motion control system acts directly on the axes, so it is usually easier to get higher performance by working in the joint space. This paper deals with the case of the joint trajectory planning.

Curve fitting is a classic problem in computer graphics and computer-aided design, and it is increasingly applied to joint trajectory planning [1–7]. The first work where the time-parameterized curve fitting is used in this application field is probably [4]. In that work, Wang uses univariate B-splines to interpolate a sequence of $\mathbb{R}^6$ points to construct the time-optimal and jerk-continuous joint trajectories of a 6-axis manipulator, respecting constraints on maximum velocity, acceleration and jerk, and concluding how this representation offers computational efficiency, flexibility and a compact format. More recently, Yang et al. [5] introduce time-parameterized quadratic B-splines to propose a certified curve fitting algorithm for G01 polylines of CNC machine ensuring the Hausdorff distance is within a certain accuracy. Following this, Huang et al. [6] propose an optimal time-jerk trajectory planning technique for robotic manipulators, which uses 5th-order B-splines to interpolate a series of points in the joint space and perform a two-objective optimization using a non-dominated sorting genetic algorithm (NSGA-II). Lin et al. [7], then again address the curve fitting problem of G01 polylines by

* Corresponding author.
E-mail address: marco.riboli@unipr.it (M. Riboli).

https://doi.org/10.1016/j.cad.2022.103421
0010-4485/

introducing time-parameterized cubic B-splines to propose a certified algorithm which considers error constraints and dynamic constraints simultaneously.

An analysis of the state-of-art shows how the most followed approach is given by solving an equation system defined by the type of trajectory and the constraints imposed on it, leading to a "polymorphic" approach, in practice consisting of using different equations for different application cases. This way to proceed has two fundamental limitations, given by the difficulty of dealing with problems that cannot be described in a closed-form and of modifying or reusing the solution found if it can no longer rely rigidly on the same scheme. To overcome these limitations, a different approach is proposed by CamOMiLe software language (Cam Open Motion Language) [8,9], based on designing piecewise trajectories by concatenating elementary trajectory pieces defined with factory functions, such as constants, lines or sine curves. Each trajectory, or piece of it, is represented by a stream of points defined on an equidistant 1D time mesh and saved on temporary files. This simple representation allows to define a library of functions that work indiscriminately with any type of trajectory and that implements symbolic calculations, shape modeling, differentiation and integration operations. The constraints are imposed with iterative loops (while and for loops) with a numerical approach, defining reusable solutions through macro-instructions definition. CamOMiLe programming language is modular and flexible enough to be used both for cam mechanism and electronically controlled axis design, allowing to implement and eventually combine both deterministic algorithms and machine learning techniques. Even if a proprietary numerical derivation algorithm has been developed to elaborate applications of considerable complexity, the numerical approach by nature still can suffer from instabilities, especially in the initial and final sections of the trajectories.

In this work the same general concept of CamOMiLe is solved by a different approach. Instead of a sampled representation, the trajectory pieces are represented by time-parameterized curves and computer-aided design concepts are applied to develop the framework basic operations. This approach is named IPTP (Iso-Parametric Trajectory Planning), and allows to overcome the limits of the tool described above defined by numerical instabilities, providing an inherently robust solution. Specifically for this work, the multi-degree univariate B-splines are used to represent trajectory pieces in the joint space. The proposed approach allows easy off-line point-to-point motion planning and allows a general approach to the development of multi-objective algorithms for trajectories subject to combined kinematic and geometric constraints, expressed relatively in the joint and the Cartesian space. The application areas are among the most varied, including pick-and-place operations, where 3D geometric path constraints are often dictated by collision avoidance algorithms [10] and friction models between the transported part and the end-effector (often a vacuum cup or magnet). Another example is the NC machining, where it is required to minimize the deviation error from a geometric tool path, often defined in Cartesian space as a polyline via G01 code [7]. This goal must be achieved while respecting dynamic constraints, and avoid that the machine tool collides with the workpieces [11]. The parameterization adopted also allows the definition of a compact data structure suitable for real-time interpolators, while the high computational performance allows modifications of the real-time trajectory such as scaling to impose a velocity override, or of a geometric path offset to re-generate the trajectory consequent to the selection of a different tool size [12]. The theorization of the framework is then followed by the implementation of our pyIPTP library, entirely developed in C, Cython and Python3 languages. In the rest of the paper the algorithmic sections are reported in object-oriented pseudo-code adopting the dot-notation for access to the instance properties.

Further parts of this paper are organized as follows. Section 2 presents a general overview on B-spline theory by exposing the algorithms used in the IPTP framework. Section 3 presents the new method and reports a benchmark in order to demonstrate the gains in performance and robustness compared to the CamOMiLe software language. Section 4 provides an experimental case study of the time-optimal multi-segment trajectory planning (TOTP) with jerk limits and constrained kinematics of an electrical axis. This example is not intended as an exhaustive discussion of all the possible design choices that could be considered to undertake the specific problem, which is better addressed in [13–15], but is intended to illustrate the flexibility and generality of the IPTP framework that easily allows to describe and extend an already established methodology classically solved in closed-form. Finally, we conclude with a summary of our work in Section 5. The numerical derivation method implemented in the CamOMiLe compiler is briefly recovered in the Appendix.

The mathematical notations and abbreviations used in the remainder of the paper not specified in the text are summarized below.

Notation
$:=$ is "equal by definition".
$\boldsymbol{\tau}$, $(\tau_i)_{i=0}^{n-1}$, $(\tau_i)_0^{n-1}$ and $\{\tau_0, \ldots, \tau_{n-1}\}$ are various ways of describing the same n-vector.
$\boldsymbol{M}$, $(\kappa_{i,j})_{i=0,j=0}^{m-1n-1}$ are various ways of describing the same $m \times n$ matrix.
$\#\boldsymbol{\tau} :=$ Number of elements of the $\boldsymbol{\tau}$ vector.
$\#\boldsymbol{s}(x) :=$ Number of elements used to define the $\boldsymbol{s}(x)$ piecewise polynomial function.
div $:=$ Integer division.
mod $:=$ Remainder after an integer division.
$\mathcal{A}(\overline{abcd}) :=$ Area of $\overline{abcd}$.
$\widehat{bcd} :=$ Vertex c of $\overline{abcd}$.
$\leftarrow$ is assignment in algorithm sections.
Acronyms
pp $:=$ Piecewise polynomial.
B-form $:=$ B-spline form.
BB-form $:=$ Bernstein-Bézier form.
CamOMiLe $:=$ Cam Open Motion Language.
IPTP $:=$ Iso-Parametric Trajectory Planning.
TOTP $:=$ Time-Optimal Trajectory Planning.
SVAJ $:=$ Position, velocity, acceleration and jerk diagrams.

2. Univariate B-splines: a general overview

In this section a summary of the existing literature on the univariate B-splines theory is presented, in order to introduce some algorithms which will be used throughout this paper.

2.1. Univariate B-splines basis

Let $\boldsymbol{t} = (t_i)_{i=0}^{m}$ be a non-decreasing knot sequence with $t_i \leq t_{i+1}$. A B-spline of order $p+1$ is a **pp** function of degree p in the parametric variable $\tau \in \mathbb{R}$, defined, in its **B-form**, by the linear combination:

$$\boldsymbol{r}(\tau) = \sum_{i=0}^{n} B_{i,p,\boldsymbol{t}}(\tau) \cdot c_i \ , \quad t_s \leq \tau \leq t_e \ , \quad n \geq p-1 \tag{1}$$

between a set of $n+1$ coefficients $\boldsymbol{c} = (c_i)_{i=0}^{n}$, with $n = m - p - 1$, and as many basis functions $B_{i,p,\boldsymbol{t}}$, with i number of knot span. The basis functions are defined by Cox-de Boor recursion formula [16]:

$$B_{i,0,\boldsymbol{t}}(\tau) = \begin{cases} 1, & \text{if } t_i \leq \tau < t_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$B_{i,p,\boldsymbol{t}} = \frac{\tau - t_i}{t_{i+p} - t_i} \cdot B_{i,p-1,\boldsymbol{t}} + \frac{t_{i+p+1} - \tau}{t_{i+p+1} - t_{i+1}} \cdot B_{i+1,p-1,\boldsymbol{t}} \tag{3}$$

In this work we use only non-periodic and non-uniform knot vector, therefore in the form:

$$\boldsymbol{t} = \{\underbrace{t_s, \ldots, t_s}_{p+1}, t_{p+1}, \ldots, t_{m-p-1}, \underbrace{t_e, \ldots, t_e}_{p+1}\} \tag{4}$$

with at least $m = 2 \cdot p + 1$.

It is useful to add some detail for a frequently used special knot vector. For convenience, set $t_s = 0$, $t_e = 1$ and $n = p$, then the knot vector is:

$$\boldsymbol{t} = \{\underbrace{0, \ldots, 0}_{p+1}, \underbrace{1, \ldots, 1}_{p+1}\} \tag{5}$$

and the curve degenerates into the **BB-form**:

$$\boldsymbol{r}^b(\tau) = \sum_{i=0}^{p} b_{i,p}(\tau) \cdot c_i \ , \quad 0 \le t \le 1 \tag{6}$$

where the basis functions $b_{i,p}$ are the classical pth-degree Bernstein polynomial:

$$b_{i,p}(\tau) := \binom{p}{i} \cdot \tau^i (1-\tau)^{p-i} = \frac{p!}{i!\,(p-i)!} \cdot \tau^i (1-\tau)^{p-i} \tag{7}$$

2.2. B-splines basic algorithms

2.2.1. Knot insertion & removal

Knot insertion procedure starts from a known B-spline with knots $\boldsymbol{t}$ and coefficients $\boldsymbol{c}$, and calculates new coefficients of a B-spline with the same shape and knots $\hat{\boldsymbol{t}}$ obtained from $\boldsymbol{t}$ by sorted insertion (increasing) of the knot $\hat{t}$ with multiplicity r. Let $\hat{t} \in [t_k, t_{k+1})$ be a knot of initial multiplicity s in $\boldsymbol{t}$, and suppose it is to be inserted r times with $r + s \le p$, [17]:

$$c_i^r = \alpha_i^r \cdot c_i^{r-1} + (1 - \alpha_i^r) \cdot c_{i-1}^{r-1} \tag{8}$$

with

$$\alpha_i^r = \begin{cases} 1, & i \le k - p + r + 1 \\ (\hat{t} - t_i)/(t_{i+p-r+1} - t_i), & k - p + r \le i \le k - s \\ 0, & i \ge k - s + 1 \end{cases} \tag{9}$$

In the case it is necessary to insert many knots at once, the knot insertion algorithm can be called multiple times, but this is not efficient. Instead, an algorithm called knot refinement [18] is used to ensure higher performance. The definition of this algorithm does not require additional mathematics to (8) and (9) but it is only a software issue and is not presented in this paper.

Reverse operation of knot insertion is called knot removal. Let t_k be an interior knot of multiplicity s in $\boldsymbol{t}$. The general equations for computing the new coefficients for r removal of t_k are [17,19]:

$$c_i^r = \frac{c_i^{r-1} - (1 - \alpha_i) \cdot c_{i-1}^r}{\alpha_i}, \quad k - p - r + 1 \le i \le \tfrac{1}{2}(2k - p - s - r) \tag{10}$$

$$c_j^r = \frac{c_j^{r-1} - \alpha_j \cdot c_{j+1}^r}{(1 - \alpha_j)}, \quad \tfrac{1}{2}(2k - p - s + r + 1) \le j \le k - s + r - 1 \tag{11}$$

with

$$\alpha_i = \frac{t_k - t_i}{t_{i+p+r} - t_i}, \quad \alpha_j = \frac{t_k - t_{j-r+1}}{t_{j+p+1} - t_{j-r+1}} \tag{12}$$

However, it is not generally known in advance if a knot is removable without changing the shape of the curve and if it is how many times. For this reason, a tolerance value is generally used. For further details see [19].

In the case it is necessary to remove as many knots as possible, the knot removal algorithm can be called for each internal knot, but this is inefficient. In order to solve the problem, Tiller [19] suggests an optimized routine to minimizes the number of shift of coefficients and knots, which is used in our implementation.

2.2.2. Degree elevation & reduction

Let $\boldsymbol{r}(\tau)$ be a pth-degree B-spline with knots:

$$\boldsymbol{t} = \{t_0, \ldots, t_m\} = \{\underbrace{t_s, \ldots, t_s}_{p+1}, \underbrace{t_1, \ldots, t_1}_{m_1}, \ldots, \underbrace{t_{\tilde{s}}, \ldots, t_{\tilde{s}}}_{m_{\tilde{s}}}, \underbrace{t_e, \ldots, t_e}_{p+1}\} \tag{13}$$

where $\tilde{s}$ is the number of distinct internal knots $\tilde{\boldsymbol{u}}$ and the coefficients $m_1, \ldots, m_{\tilde{s}}$ are their multiplicities. The term degree elevation indicates the operation that defines knots and coefficients of a new B-spline of degree $p + 1$ of the same shape as $\boldsymbol{r}(\tau)$. The new knots $\hat{\boldsymbol{t}}$ can be written as:

$$\hat{\boldsymbol{t}} = \{\hat{t}_0, \ldots, \hat{t}_{\bar{m}}\} = \{\underbrace{t_s, \ldots, t_s}_{p+2}, \underbrace{t_1, \ldots, t_1}_{m_1+1}, \ldots, \underbrace{t_{\tilde{s}}, \ldots, t_{\tilde{s}}}_{m_{\tilde{s}}+1}, \underbrace{t_e, \ldots, t_e}_{p+2}\} \tag{14}$$

with maximum index $\bar{m} = m + \tilde{s} + 2$. Moreover, (14) allows to keep the same continuity as $\boldsymbol{r}(\tau)$. In [20] an efficient algorithm to get the $\bar{n} + 1$ coefficients, with $\bar{n} = n + \tilde{s} + 1$, is shown. This approach performs the following operations on-the-fly: extracts the ith Bézier segment (see [17] for curve decomposition algorithm), increases its degree and then eliminates unnecessary knots. The calculation of the degree-elevated Bézier coefficients after r degree elevations can be performed with:

$${}^r c_i = \sum_{j=max(0,i-r)}^{min(p,i)} \frac{\binom{p}{j}\binom{r}{i-j} c_j}{\binom{p+r}{i}}, \quad i = 0, \ldots, p + r \tag{15}$$

While it is always possible to elevate the B-spline degree, a curve may not be reducible. Let ${}^1\boldsymbol{r}(\tau)$ be a pth-degree B-spline with knots defined by (13). ${}^1\boldsymbol{r}(\tau)$ is degree reducible if it has an "accurate" representation [17]:

$${}^1\boldsymbol{r}(\tau) = \boldsymbol{r}(\tau) = \sum_{i=0}^{n-\tilde{s}-1} B_{i,p-1,\bar{\boldsymbol{t}}}(\tau) \cdot c_i \tag{16}$$

with knots

$$\bar{\boldsymbol{t}} = \{\hat{t}_0, \ldots, \hat{t}_{\bar{m}}\} = \{\underbrace{t_s, \ldots, t_s}_{p}, \underbrace{t_1, \ldots, t_1}_{m_1-1}, \ldots, \underbrace{t_{\tilde{s}}, \ldots, t_{\tilde{s}}}_{m_{\tilde{s}}-1}, \underbrace{t_e, \ldots, t_e}_{p}\} \tag{17}$$

The algorithm used for the degree reduction of B-splines follows the same three-step strategy used for degree elevation, where the degree elevation of the ith Bézier curve is replaced with a degree reduction. The literature provides several degree reduction algorithms for Bézier curves [21,22]. We adopt the algorithm proposed in [23]. Let

$$r = (p - 1) \operatorname{div} 2 \tag{18}$$

$$\alpha_i = i/p \tag{19}$$

$${}^1\boldsymbol{r}^b(\tau) = \sum_{i=0}^{p} b_{i,p}(\tau) \cdot {}^1 c_i \tag{20}$$

$$\boldsymbol{r}^b(\tau) = \sum_{i=0}^{p-1} b_{i,p-1}(\tau) \cdot c_i \tag{21}$$

then, if $p \bmod 2 = 0$ (even):

$$c_0 = {}^1 c_0, \quad c_{p-1} = {}^1 c_p \tag{22}$$

$$c_i = \frac{{}^1 c_i - \alpha_i \cdot c_{i-1}}{1 - \alpha_i}, \quad \text{for } i = 1, \ldots, r \tag{23}$$

$$c_i = \frac{{}^1 c_{i+1} - (1 - \alpha_{i+1}) \cdot c_{i+1}}{\alpha_{i+1}}, \quad \text{for } i = p - 2, \ldots, r + 1 \tag{24}$$

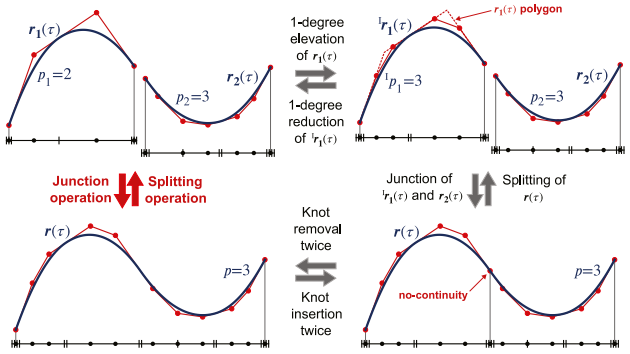


Fig. 1. Generic pattern for B-spline junction and splitting.

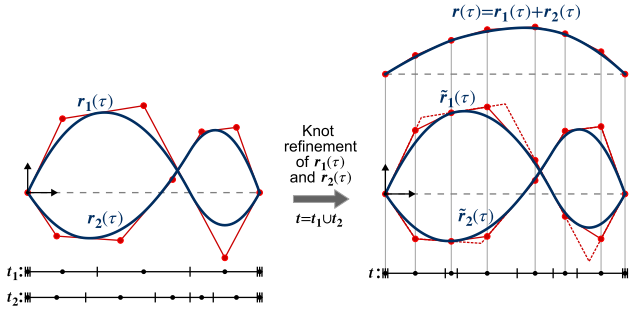


Fig. 2. Sum operator between two B-splines.

else, if $p \bmod 2 = 1$ (odd):

$$c_0 = {}^1c_0, \quad c_{p-1} = {}^1c_p \tag{25}$$

$$c_i = \frac{{}^1c_i - \alpha_i \cdot c_{i-1}}{1-\alpha_i}, \quad \text{for } i = 1, \dots, r-1 \tag{26}$$

$$c_r = \frac{1}{2}\left(\frac{{}^1c_r - \alpha_r \cdot c_{r-1}}{1-\alpha_r} + \frac{{}^1c_{r+1} - (1-\alpha_{r+1}) \cdot c_{r+1}}{\alpha_{r+1}}\right) \tag{27}$$

$$c_i = \frac{{}^1c_{i+1} - (1-\alpha_{i+1}) \cdot c_{i+1}}{\alpha_{i+1}}, \quad \text{for } i = p-2, \dots, r+1 \tag{28}$$

2.2.3. Junction and splitting

We now consider two applications of the algorithms presented in 2.2.1 and 2.2.2.

In our software, the piecewise trajectory is performed by concatenation of the single trajectory pieces. A generic pattern to perform this operation between two B-splines $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ is shown in Fig. 1. Initially, **pp** functions may not have the same degree. In this case the curves must be made compatible. Since a B-spline is not always reducible, the more generic case sees the lower degree curve elevated with (15) until the degree $p = \max(p_1, p_2)$. In Fig. 1, $\boldsymbol{r_1}$ is elevated one time and ${}^1\boldsymbol{r}_1$ is obtained. The concatenation of the two B-splines is now easily solved. Let

$${}^1\boldsymbol{t}_1 = \{\underbrace{{}^1t_{1,s}, \dots, {}^1t_{1,s}}_{p+1}, {}^1t_{1,p+1}, \dots, {}^1t_{1,n_1}, \underbrace{{}^1t_{1,e}, \dots, {}^1t_{1,e}}_{p+1}\} \tag{29}$$

$$\boldsymbol{t_2} = \{\underbrace{t_{2,s}, \dots, t_{2,s}}_{p+1}, t_{2,\,p+1}, \dots, t_{2,\,n_2}, \underbrace{t_{2,e}, \dots, t_{2,e}}_{p+1}\} \tag{30}$$

$$\Delta t = {}^1t_{1,e} - t_{2,s} \tag{31}$$

$${}^1\boldsymbol{c}_1 = \{{}^1c_{1,0}, \dots, {}^1c_{1,n_1}\} \tag{32}$$

$$\boldsymbol{c_2} = \{c_{2,0}, \dots, c_{2,n_2}\} \tag{33}$$

The B-spline $\boldsymbol{r}(\tau)$ given by the concatenation of ${}^1\boldsymbol{r}_1$ and $\boldsymbol{r_2}$ is defined by:

$$\begin{aligned}\boldsymbol{t} = \{&\underbrace{{}^1t_{1,s}, \dots, {}^1t_{1,s}}_{p+1}, {}^1t_{1,p+1}, \dots, {}^1t_{1,n_1}, \underbrace{{}^1t_{1,e}, \dots, {}^1t_{1,e}}_{p+1}, \dots,\\ &(t_{2,\,p+1} + \Delta t), \dots, (t_{2,\,n_2} + \Delta t),\\ &\underbrace{(t_{2,e} + \Delta t), \dots, (t_{2,e} + \Delta t)}_{p+1}\}\end{aligned} \tag{34}$$

$$\boldsymbol{c} = \{{}^1c_{1,0}, \dots, {}^1c_{1,n_1}, c_{2,0}, \dots, c_{2,n_2}\} \tag{35}$$

where in ${}^1t_{1,e}$ the B-spline is no-continuous. If ${}^1c_{1,n_1} = c_{2,0}$, then ${}^1t_{1,e}$ is removable at least once. In general, (10) and (11) are then used to remove the knot as many times as possible, obtaining the right continuity in the junction point.

Fig. 1 also shows how this pattern is invertible using (8) and degree reduction algorithms [23]. In general, to split a B-spline $\boldsymbol{r}(\tau)$ into knot $\hat{t} \in (t_s, t_e)$ with initial multiplicity s, $\hat{t}$ must be inserted $p+1-s$ times.

2.2.4. Differentiation and integration

Let $\boldsymbol{r}(\tau)$ be a pth-degree B-spline described from (1). The rth derivative of $\boldsymbol{r}(\tau)$ is computed by [16]:

$$D^r\boldsymbol{r}(\tau) = \textstyle\sum_{i=0}^{n-r} B_{i,p-r,\boldsymbol{t}}(\tau) \cdot c_i^r \tag{36}$$

with

$$c_i^r = \frac{p-r+1}{t_{i+r+1} - t_{i+k}}\left(c_{i+1}^{r-1} - c_i^{r-1}\right), \quad \text{with } r > 0 \tag{37}$$

and knots:

$$\boldsymbol{t}^r = \{\underbrace{t_s, \dots, t_s}_{p-r+1}, t_{p+1}, \dots, t_{m-p-1}, \underbrace{t_e, \dots, t_e}_{p-r+1}\} \tag{38}$$

The antiderivative of $\boldsymbol{r}(\tau)$ is instead computed by [16]:

$$D^{-1}\boldsymbol{r}(\tau) = \textstyle\sum_{i=0}^{n+1} B_{i,p+1,\boldsymbol{t}}(\tau) \cdot c_i^{-1} \tag{39}$$

with

$$c_i^{-1} = \sum_{j=1}^{i} c_j \cdot \frac{t_{j+p+1} - t_j}{p+1}, \quad \text{for } i = 0, \dots, n+1 \tag{40}$$

and new knots:

$$\boldsymbol{t}^{-1} = \{\underbrace{t_s, \dots, t_s}_{p+2}, t_{p+1}, \dots, t_{m-p-1}, \underbrace{t_e, \dots, t_e}_{p+2}\} \tag{41}$$

Then, (39) is iterated k times to obtain the kth antiderivative.

2.3. Symbolic calculation

Let $\boldsymbol{r_1}(\tau)$ and $\boldsymbol{r_2}(\tau)$ be two B-splines of the same degree $p_1 = p_2 = p$ and common domain $\boldsymbol{t}_1 = \boldsymbol{t}_2 = \boldsymbol{t}$, the simple formula for adding the two curves is [24]:

$$\begin{aligned}\boldsymbol{r}(\tau) &= \boldsymbol{r_1}(\tau) + \boldsymbol{r_2}(\tau)\\ &= \textstyle\sum_{i=0}^{n} B_{i,p,\boldsymbol{t}}(\tau) \cdot c_{i,1} + \sum_{i=0}^{n} B_{i,p,\boldsymbol{t}}(\tau) \cdot c_{i,2}\\ &= \textstyle\sum_{i=0}^{n} B_{i,p,\boldsymbol{t}}(\tau) \cdot \left(c_{i,1} + c_{i,2}\right)\end{aligned} \tag{42}$$

Hence, the unknown coefficients $\boldsymbol{c}$ are defined by $c_i = c_{i,1} + c_{i,2}$, and the problem is therefore reduced to make the curves compatible. Assuming that $t_{s,1} = t_{s,2}$ and $t_{e,1} = t_{e,2}$, knot refinement and degree elevation algorithms are performed to make the curves compatible [25]. If $t_{s,1} \neq t_{s,2}$ or $t_{e,1} \neq t_{e,2}$, it is sufficient that $\boldsymbol{t}_1 \cap \boldsymbol{t}_2 \neq 0$ and it is always possible to go back to the previous case using splitting operations. When the operation is completed, the previously splitted curve pieces can be joined, and the problem is solved (see Fig. 2).

Finding products of two B-splines is more difficult than the sum computation, and finds different solutions in the literature.

These are mainly divided into direct and indirect approach, where indirect methods leave the **B-form** to calculate the product of each set of polynomial pieces or use interpolations after sampling. Indirect methods are classically more efficient and usually preferred [26]. In this paper the indirect method proposed by Piegl and Tiller [27] is used.

Let $\boldsymbol{r_1}(t)$ and $\boldsymbol{r_2}(t)$ be two B-splines of degree p_1 and p_2 respectively, with knots:

$$\boldsymbol{t_1} = \{\underbrace{t_s, \dots, t_s}_{p_1+1}, t_{1,\,p_1+1}, \dots, t_{1,\,n_1}, \underbrace{t_e, \dots, t_e}_{p_1+1}\} \tag{43}$$

$$\boldsymbol{t_2} = \{\underbrace{t_s, \dots, t_s}_{p_2+1}, t_{2,\,p_2+1}, \dots, t_{2,\,n_2}, \underbrace{t_e, \dots, t_e}_{p_2+1}\} \tag{44}$$

The unknowns are the coefficients $\boldsymbol{c}$ of the product curve of degree $p_1 + p_2$:

$$\boldsymbol{r}(\tau) = \boldsymbol{r_1}(\tau) \cdot \boldsymbol{r_2}(\tau) = \textstyle\sum_{i=0}^{n} B_{i,p+q,\boldsymbol{t}}(\tau) \cdot c_i \tag{45}$$

with

$$\boldsymbol{t} = \{\underbrace{t_s, \dots, t_s}_{p_1+p_2+1}, t_{p_1+p_2+1}, \dots, t_n, \underbrace{t_e, \dots, t_e}_{p_1+p_2+1}\} \tag{46}$$

where distinct internal knots are defined by $\check{\boldsymbol{u}} = \check{\boldsymbol{u}}_{\boldsymbol{1}} \cup \check{\boldsymbol{u}}_{\boldsymbol{2}}$ with multiplicities:

$$m_i = \begin{cases} p_2 + m_{1,i}, & \text{if } m_{2,i} = 0 \\ p_1 + m_{2,i}, & \text{if } m_{1,i} = 0 \\ \max(p_2 + m_{1,i},\ p_1 + m_{2,i}), & \text{otherwise} \end{cases} \tag{47}$$

The coefficients $\boldsymbol{c}$ are computed by converting the curves from **B-form** to piecewise **BB-form** with knot insertion algorithm, performs the Bézier multiplication [24,28], then returns to the **B-form** with knot removal algorithm. The problem is therefore reduced to calculating the product of two Bézier curves.

Let $\boldsymbol{r}_{\boldsymbol{1}}^{b}(\tau)$ and $\boldsymbol{r}_{\boldsymbol{2}}^{b}(\tau)$ be two Bézier curves of coefficients $\boldsymbol{c}_1 = (c_{1,i})_{i=0}^{p_1}$ and $\boldsymbol{c}_2 = (c_{2,i})_{i=0}^{p_2}$, and degree p_1 and p_2, respectively. The product curve of degree $p = p_1 + p_2$ is defined by:

$$\boldsymbol{r}^{b}(\tau) = \textstyle\sum_{i=0}^{p} b_{i,p}(\tau) \cdot c_i \tag{48}$$

with

$$c_i = \sum_{j=\max(0,i-p_2)}^{\min(p_1,i)} \frac{\binom{p_1}{j}\binom{p_2}{i-j} c_{1,j} \cdot c_{2,i-j}}{\binom{p}{i}}\ , \quad i = 0, \dots, p \tag{49}$$

Note that if $c_{2,i} = 1$ then (49) describes the same coefficients as (15) and the B-spline product produces a degree elevation.

2.4. B-splines curve fitting

The curve fitting problems of a known function can be divided into two main categories. If the function to be approximated is polynomial or piecewise polynomial, the representation by univariate B-splines is exact. Otherwise, for example with trigonometric functions, the solution is approximated and computed with interpolation methods. In case the function to be approximated is not known and only sampled data are available (often subject to noise), approximation techniques often based on least squares are used, but are not exposed in this paper.

Any polynomial of degree p defined by $p + 1$ coefficients $\boldsymbol{a} = (a_k)_0^p$ in its **pp-form**:

$$\boldsymbol{p}(x) = \textstyle\sum_{i=0}^{p} x^i \cdot a_i = a_0 + x \cdot a_1 + x^2 \cdot a_2 + \dots + x^p \cdot a_p \tag{50}$$

can be described also in its **BB-form**, hence again with (6). To this end, let us describe the generic Bézier curve using the matrix $\boldsymbol{M} = (\kappa_{i,j})_{i=0,\,j=0}^{p\ \ p}$:

$$\begin{aligned} \boldsymbol{r}^{b}(x) &= \textstyle\sum_{i=0}^{p} b_{i,p}(x) \cdot c_i \\ &= \begin{bmatrix} 1, x, x^2, \dots, x^p \end{bmatrix} \cdot \begin{bmatrix} \kappa_{0,0} & 0 & \cdots & 0 \\ \kappa_{1,0} & \kappa_{1,1} & & \vdots \\ \vdots & \vdots & \ddots & 0 \\ \kappa_{p,0} & \kappa_{p,1} & \cdots & \kappa_{p,p} \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_p \end{bmatrix} \\ &= \boldsymbol{x} \cdot \boldsymbol{M} \cdot \boldsymbol{c} \end{aligned} \tag{51}$$

with

$$\kappa_{i,j} = \tbinom{p}{i} \cdot \tbinom{i}{j} \cdot (-1)^{i-j}\ , \quad \text{with } \begin{cases} i = 0, \dots, p \\ j = 0, \dots, i \end{cases} \tag{52}$$

By comparing (50) with (51), the unknown coefficients c_i can be found by solving the linear system $\boldsymbol{M} \cdot \boldsymbol{c} = \boldsymbol{a}$. The matrix $\boldsymbol{M}$ is lower triangular and an appropriate method must be used (in our software on-fly forward substitution is adopted). Note that the coefficients so established are related to a normalized domain $\boldsymbol{t} = \{0, \dots, 0, 1, \dots, 1\}$. This is often desired; the polynomial trajectories are classically reported in the handbooks in the form of coefficients relating to a dimensionless domain and unitary lift. If it is not desired, it is easy to fix it by redefining the coefficients:

$$c_i = c_i \cdot \beta^i\ , \quad \beta = t_e - t_s \tag{53}$$

with knots $\boldsymbol{t} = \{t_s, \dots, t_s, t_e, \dots, t_e\}$.

In the case of classic or Hermite splines, they are often given in **pp-form**:

$$\boldsymbol{s}(x) := \boldsymbol{p}_j(x)\ , \quad x \in [\xi_j,\ \xi_{j+1}], \quad j = 0, \dots, l \tag{54}$$

with $\boldsymbol{p}_j(x)$ defined by (50) and which presupposes the definition of break sequence $\boldsymbol{\xi} = (\xi_j)_0^l$ and a coefficients matrix $\boldsymbol{A} = (a_{i,j})_{i=0,\,j=0}^{p\ \ l}$ with:

$$a_{i,j} = D^{i-1}\boldsymbol{p}(\xi_j)\ , \quad i = 0, \dots, p\ , \quad j = 0, \dots, l \tag{55}$$

One approach for converting a **pp** function from **pp-form** to **B-form** would be to convert each $\boldsymbol{p}_j(x)$ into its own **BB-form**, and concatenate them by removing knots in the joints according to the continuity conditions, but it does not seem to be the most efficient approach. A more efficient approach is reported in [29, p. 10–4].

If the trajectory piece $f(t)$ can only be approximated by a B-spline, the fit of the curve is performed through interpolation techniques and the degree p is arbitrary and not defined by $f(t)$. Let $\boldsymbol{\tau} = (\tau_k)_{k=0}^{n}$ be a non-decreasing interpolation abscissas, with $\tau_i \le \tau_{i+1}$, and $\boldsymbol{q} = (f(\tau_k))_{k=0}^{n} = (q_k)_{k=0}^{n}$ be the interpolating ordinates associated with $\boldsymbol{\tau}$. The coefficients $\boldsymbol{c}$ of the B-spline $\boldsymbol{r}(t)$ which interpolates the points $\{\tau_k, q_k\}$ are computed by solving the system:

$$q_k = \boldsymbol{r}(\tau_k) = \textstyle\sum_{i=0}^{n} B_{i,p,\boldsymbol{t}}(\tau_k) \cdot c_i \tag{56}$$

The knots $\boldsymbol{t}$ must be chosen in order to fulfill the Schoenberg–Whitney condition [16]:

$$t_i < \tau_i < t_{i+p+1}, \quad i = 1, \dots, m-1 \tag{57}$$

To this end, the knots can be select with the formula [17]:

$$\begin{aligned} &t_0 = \dots = t_p = t_s\ , \quad t_{m-p} = \dots = t_m = t_e \\ &t_{j+p} = (1/p) \textstyle\sum_{i=j}^{j+p-1} \tau_i\ , \quad j = 1, \dots, n-p \end{aligned} \tag{58}$$

An extremely general algorithm is reported in [30], which adds to system (56) the $\iota + \kappa$ conditions:

$$d_{\kappa}^{\jmath} = D^{\jmath}\boldsymbol{r}(\bar{\tau}) = \textstyle\sum_{i=0}^{n} D^{\jmath} B_{i,p,\boldsymbol{t}}(\bar{\tau}) \cdot c_i \tag{59}$$

with

$$\begin{cases} j = 0, \dots, \kappa & \text{if } \bar{\tau} = t_s \\ j = 0, \dots, \iota & \text{if } \bar{\tau} = t_e \end{cases} \tag{60}$$

in order to constrain up to the κth and ιth derivatives at the starting and ending points of the curve, respectively. In matrix form (substituting for simplicity $D^j B_{i,p,\boldsymbol{t}}$ to B_i^j):

$$\begin{bmatrix} 1 & & & & & & & & \\ B_0^1(t_s) & B_1^1(t_s) & & & & & & & \\ \vdots & \ddots & & & & & & & \\ B_0^\kappa(t_s) & \cdots & B_k^\kappa(t_s) & & & & & & \\ & & B_0(t_1) & \cdots & B_p(t_1) & & & & \\ & & \ddots & & \ddots & & & & \\ & & & B_{m-p}(t_{n-1}) & \cdots & B_m(t_{n-1}) & & & \\ & & & & B_{m-l}^\iota(t_e) & \cdots & B_m^\iota(t_e) & & \\ & & & & \ddots & & \vdots & & \\ & & & & & B_{m-1}^1(t_e) & B_m^1(t_e) & & \\ & & & & & & 1 & & \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \\ c_m \end{bmatrix} = \begin{bmatrix} q_0 \\ d_s^1 \\ \vdots \\ d_s^\kappa \\ q_1 \\ \vdots \\ q_{n-1} \\ d_e^\iota \\ \vdots \\ d_e^1 \\ q_n \end{bmatrix} \tag{61}$$

The extra knots must be inserted in order to obtain a system (61) banded, and diagonally dominant in the part where the basic non-vanishing functions are placed. To this end, the extra knots can be inserted in the first and last spans. A way to place extra knots that ensures numerical stability is presented in [30].

3. Trajectory planning in the IPTP framework

IPTP defines a new paradigm in joint trajectory planning that generalizes traditional solutions and provides a set of tools to model trajectories similar to geometric CAD software. The time-parameterized representation allows to define libraries of functions which allow the designer to plan the trajectory using both closed-form solutions and numerical approaches. Since joint trajectories are bijective functions, we have adopted univariate B-splines to represent trajectories as they offer high computational efficiency and a compact format, as well as being considered a standard in computer-aided design and computer graphics.

In order to contextualize what is stated in our open-source software, in Fig. 3 we report a simplified architecture of the pyIPTP library. The application places the B-splines library in the lowest level, which implements the algorithms reported in Section 2. This section focuses instead on the middle and high level, where the framework is built in order to allow a simple use by means of the object-oriented paradigm. With reference to Fig. 4, the Cython tool [31] is used to define a built-in extension Trajectory1D in order to represent a single piece of trajectory

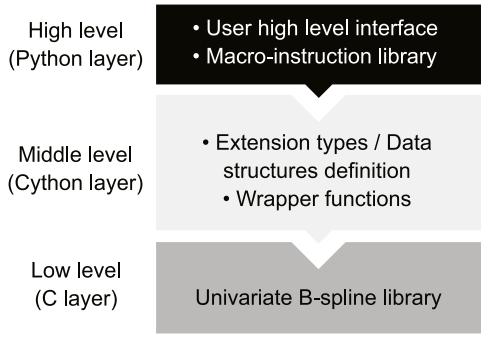


Fig. 3. Three-level architecture of the pyIPTP library.

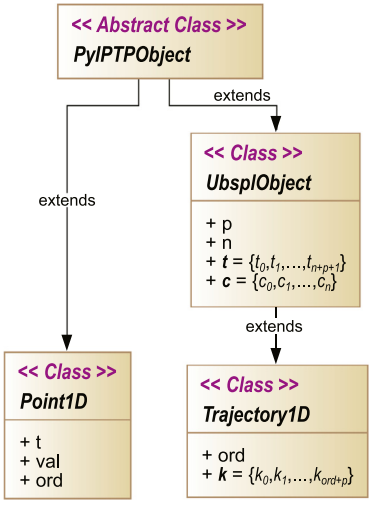
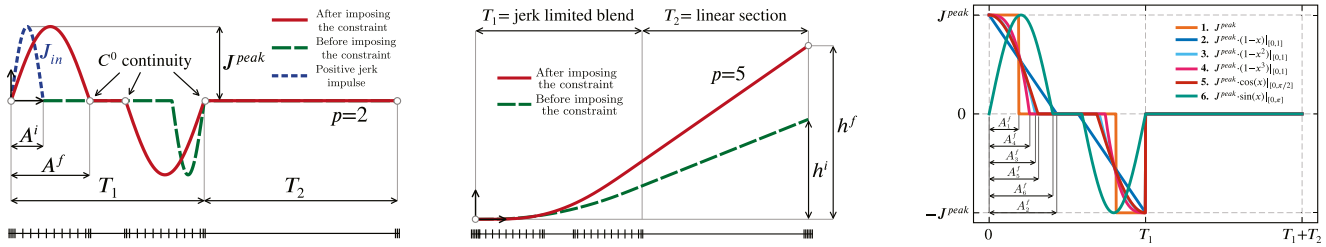


Fig. 4. Inheritance diagram of the pyIPTP library.

and wrap the functions of the low-level numeric library. This architecture allows for C-like computational performance, while offering the user all the advantages of an interpreted language like Python3.

Trajectory1D class extends the superclass UbsplObject adding specific properties to describe trajectories, such as the derivation order *ord* and the integration constants $\boldsymbol{k} = (k_i)_{i=0}^{ord+p}$, the latter necessary to preserve the initial conditions of the trajectory piece in the integration operations. The UbsplObject class it is used instead to represent instances of purely geometric B-spline curves, thus storing all the characteristic parameters of the curve. This superclass is necessary for the representation of physical quantities that are not a displacement or any of its time derivatives, e.g. a power characteristic obtained by multiplying velocity and acceleration pieces. In this case, the UbsplObject class can be further extended to define an appropriate object class to handle that physical quantity. The UbsplObject class in turn extends the abstract class PyIPTPObject, used for all the framework objects and which implements some utilities. Another object class that is useful to mention is Point1D, which stores a space 1D point *val* corresponding to the time abscissa *t* and for a specific derivation order *ord*. For example, a Point1D instance is useful for storing a constraint to be set on the trajectory.

In the framework, the designer creates Trajectory1D objects using *Factory functions*, which define elementary pieces, such as constants, lines, sines and polynomials. In the case of approximate solutions that use interpolation techniques (e.g. trigonometric pieces), the interpolation grid should be set in the light of a



(a) Jerk. (b) Displacement. (c) Results for different J_{in}.

Fig. 5. Schematic representation of JerkLimRamp Macro-instruction (Algorithm 2).

convergence analysis up to the rth derivative, with r evaluated for the specific application. On the other hand, the designer usually desires to start planning the trajectory from high derivatives, helping to avoid instability. The piecewise trajectory is composed by the concatenation of several elementary pieces, which can be modeled and modified with *Basic functions* that implement shape modeling, symbolic calculation, derivation and integration operations. Without demanding completeness, we report in Table 1 some of the functions provided by the current release of the pyIPTP Python3 library.

From the framework logic point of view, trajectories can be constrained in many ways, and most of the time it is convenient to define a sequence of Basic functions as a *Macro-instruction* that can be reused in multiple applications. Some examples of Macro-instructions are analyzed in the next section.

3.1. Macro-instructions

A sequence of operations used to impose one or more constraints on the trajectory is named Macro-instruction (or more simply Macro). They can be divided into two main categories; the first defines a closed-form solution, while the latter imposes the constraints with a numerical approach by adopting a design tolerance.

A simple example of a Macro that imposes the constraint exactly is presented in Algorithm 1. Let Q_{in} be a Trajectory1D object which represents a trajectory piece of any derivation order, and P a Point1D object which defines the constraint to be set. SetValueAm Macro (Algorithm 1) first calculates the vertical scale factor α to set the constraint in the correct derivation order $P.ord$, then scales Q_{in} vertically of α to obtain Q_{out}. Note that, let $\boldsymbol{q}_{out}(\tau)$ be the B-spline that defines the trajectory piece Q_{out}, for $P.val \to 0$, it follows that $\alpha \to 0$ and $\boldsymbol{q}_{out}(\tau) \to 0$, $\forall\tau$. Algorithm 1 cannot therefore be used to set a null constraint, or constraints close to zero.

Algorithm 1 SetValueAm Macro-instruction

Input:

1. Trajectory1D object Q_{in}.
2. Point1D object P.

Output:

1. Trajectory1D object Q_{out}.

Algorithm:

$\Delta \leftarrow (P.ord - Q_{in}.ord)$
Compute $D^{\Delta}Q_{in}$ using (36) or (39)
Evaluate $D^{\Delta}Q_{in}$ in $P.t$ and save the value in $\bar{h}$
$\alpha \leftarrow P.val/\bar{h}$ ▷ compute the scale factor
$Q_{out} \leftarrow Q_{in}$
Vertically scale Q_{out} of $K = \alpha$ using (71)
return Q_{out}

Algorithm 2 instead reports a Macro to define a jerk-limited ramp of total lift h and maximum jerk value J^{peak}. In this case the constraints are imposed to the trajectory piece numerically through an iterative loop using a design tolerance TOL. The algorithm requires a positive jerk impulse J_{in} defined by the B-spline $r_{in}(\tau) = J^{peak} \cdot \bar{r}_{in}(\tau)$, where $\bar{r}_{in}(\tau) : [0, \bar{\tau}] \to \mathbb{R}$, with $\bar{\tau}$ a generic constant real value, and $\|\bar{r}_{in}(\tau)\|_{\infty} = 1$. Algorithm 2 defines a blend of period T_1 followed by a piece of linear trajectory of period T_2 (see Fig. 5(b)). The blend is defined by concatenation of J_{in}, a piece of null jerk and a negative jerk impulse obtained by mirroring and negating J_{in}. With reference to Fig. 5(a), the lift constraint h is then set by horizontally scaling the two impulses towards the center of the blend, and decreasing the width of the central piece to maintain an overall period of T_1. The peak value of the jerk J^{peak} is therefore preserved during the iteration and is defined by the amplitude of J_{in}. This solution also allows to parameterize the algorithm on the positive impulse J_{in}. In this regard, Fig. 5(c) reports the results for different input impulses. Note that, let A be the impulse width of J_{in}, a limit condition $2 \cdot A = T_1$ establishes the maximum velocity that the blend of length T_1 can impose on the next linear piece.

Algorithm 2 JerkLimRamp Macro-instruction

Input:

1. Trajectory1D object J_{in}.
2. Total lift h.
3. Periods T_1 and T_2 according to Fig. 5.
4. Tolerance TOL.
5. Incremental step ε and step ratio ϕ.

Output:

1. Trajectory1D object J_{out}.

Algorithm:

$A \leftarrow \|J_{in}.\boldsymbol{t}[0] - J_{in}.\boldsymbol{t}[-1]\|$ ▷ initialize pulse width parameter
$J_1 \leftarrow J_{in}$
Initialize a piece of null trajectory J_2 with (62) on domain [0..1]
Mirror J_1 using (66) and save in the new instance J_3
Vertically scale J_3 of $K = -1$ using (71)
repeat
  repeat
    $A \leftarrow A + \varepsilon$
    assert $2 \cdot A \leq T_1$ ▷ limit condition
    Adapt J_1 in the new domain $[0..A]$ with (69)
    Adapt J_2 in the new domain $[A..T_1 - A]$ with (69)
    Adapt J_3 in the new domain $[T_1 - A..T_1]$ with (69)
    Adapt J_4 in the new domain $[T_1..T_1 + T_2]$ with (69)
    Concatenate J_1, J_2, J_3, J_4 with the pattern shown in Fig. 1 and save the result in J_{out}
    Compute $D^{-3}J_{out}$ using (39)
    Evaluate $D^{-3}J_{out}$ in $T_1 + T_2$ and save the value in $\bar{h}$
  until $(\bar{h} < h)$
  $A \leftarrow A - \varepsilon$; $\varepsilon \leftarrow \varepsilon \cdot \phi$
until $(\|\bar{h} - h\| > TOL)$
return J_{out}

Table 1
Functions provided in the actual release of pyIPTP Python3 library.

Factory functions			
Name	In		Description
constant	t_s, t_e, ord, K		Returns a new Trajectory1D object of order *ord* represented by a Bézier curve of degree $p = 0$, a single coefficient $\boldsymbol{c} = \{K\}$ and knots $\boldsymbol{t} = \{t_s, t_e\}$: $\boldsymbol{r}^b(\tau) = b_{0,0}(\tau) \cdot K = K$ (62) "zeros" and "ones" are two aliases for $K = 1$ and $K = 0$, respectively.
lin1p	$t_s, t_e, ord, \{\tau, q\}, m$		Returns a new Trajectory1D object of order *ord* represented by a Bézier curve of degree $p = 1$. Knots and coefficients are calculated as follows: $c_i = m \cdot (t_s + i\beta) + (q - m \cdot \tau)\ , \quad i = 1, 2\ , \quad \beta = t_e - t_s\ , \quad \boldsymbol{t} = \{t_s, t_s, t_e, t_e\}$ (63)
lin2p	$t_s, t_e, ord, \{\tau_1, q_1\}, \{\tau_2, q_2\}$		First calculate $m = (q_2 - \tau_2)/(q_1 - \tau_1)$, then executes "lin1p" using $\{\tau_1, q_1\}$ and m.
fromdata	$\boldsymbol{\tau} = (\tau_k)_{k=0}^n, \boldsymbol{q} = (q_k)_{k=0}^n, p,$ *ord*		Returns a new Trajectory1D object of order *ord* represented by a B-spline curve of degree p obtained by solving (61) and setting the relation: $\iota = \kappa = p - 1$ (64)
sintraj	$\boldsymbol{\tau} = (\tau_k)_{k=0}^n, p, ord, A, \phi, K$		Computes the vector $\boldsymbol{q} = (q_k)_{k=0}^n$ according the relation: $q_k = A \cdot \sin(K \cdot \tau_k + \phi)$ (65) and executes "fromdata" factory function.
costraj	$\boldsymbol{\tau} = (\tau_k)_{k=0}^n, p, ord, A, \phi, K$		Executes "sintraj" with the ϕ phase incremented by $\pi/2$.
polytraj	$\boldsymbol{a} = (a_k)_{k=0}^p, t_s, t_e, ord$		Returns a new Trajectory1D object of order *ord* represented by a B-spline curve of degree p and coefficients $\boldsymbol{c}$ computed by solving the linear system $\boldsymbol{M} \cdot \boldsymbol{c} = \boldsymbol{a}$, with $\boldsymbol{M}$ defined by (51).
bspltraj	$\boldsymbol{t} = (t_i)_{i=0}^m, \boldsymbol{c} = (c_i)_{i=0}^n, p, ord$		Returns a new Trajectory1D object of order *ord* represented by a B-spline curve of degree p, coefficients $\boldsymbol{c}$ and knots $\boldsymbol{t}$.
spltraj	$\boldsymbol{A} = (a_{i,j})_{i=0,\,j=0}^{p\quad l}, \boldsymbol{\xi} = (\xi_j)_0^l,$ *ord*		Returns a new Trajectory1D object of order *ord* represented by a B-spline curve of degree p, obtained by conversion from **pp-form** to B-form.
Basic functions			
Name	In	Out	Description
join	$Q_{in\,1}, Q_{in\,2}$	Q_{out}	Concatenates two trajectory pieces ensuring correct continuity at the junction point. A generic pattern using the (10), (11) and (15) is represented in Fig. 1.
split	$Q_{in}, \hat{t}$	$Q_{out\,1}, Q_{out\,2}$	Splits a trajectory piece Q_{in} with domain $[t_s..t_e]$ in the abscissa $\hat{t} \in [t_s..t_e]$. A generic pattern using the (8), (22)–(28) is represented in Fig. 1.
mirror	Q_{in}	Q_{out}	Let $\boldsymbol{r}(\tau)$ be the B-spline describing the trajectory to be mirrored, the calculation of the parameters of the mirrored curve $\bar{\boldsymbol{r}}(\tau)$ with $\tau \in [\bar{t}_s, \bar{t}_e]$ is processed with the formulas: $\bar{c}_i = c_{n-i}, \quad \text{with } i = 0, \ldots, n$ $\bar{t}_s = t_e, \quad \bar{t}_e = 2t_e - t_s$ (66) $\bar{t}_j = \bar{t}_{j-1} + t_e - t_{n-j}, \quad \text{with } j = p+1, \ldots, n$
hshift	Q_{in}, K	–	Translates the trajectory horizontally by calculating the new knots $\bar{\boldsymbol{t}}$ in-place with: $\bar{t}_i = t_i + K$ (67)
hscale	Q_{in}, K	–	Scales the trajectory horizontally by calculating the new knots $\bar{\boldsymbol{t}}$ in-place with: $\bar{t}_i = t_i \cdot K$ (68)
adapt	$Q_{in}, \bar{\boldsymbol{t}}$	–	Adapts the trajectory in a new domain $\bar{\boldsymbol{t}} = [\bar{t}_s, \bar{t}_e]$. Let $\boldsymbol{r}(\tau)$, with $\tau \in Q_{in}.\boldsymbol{t}$, be the B-spline associated with Q_{in}, the problem is solved by calculating the new knots in-place with: $\bar{t}_i = t_i \cdot k - t_s \cdot (k-1) + \Delta t$ (69) where $k = (t_e - t_s)/(\bar{t}_e - \bar{t}_s)$ and $\Delta t = \bar{t}_s - t_s$.
vshift	Q_{in}, K	–	Translates the trajectory vertically by calculating the new coefficients $\bar{\boldsymbol{c}}$ in-place with: $\bar{c}_i = c_i + K$ (70)
vscale	Q_{in}, K	–	Scales the trajectory vertically by calculating the new coefficients $\bar{\boldsymbol{c}}$ in-place with: $\bar{c}_i = c_i \cdot K$ (71)
der	Q_{in}, ν	Q_{out}	Compute $D^\nu Q_{in}$ using (36) and save the result in Q_{out}.
antider	Q_{in}, ν	Q_{out}	Compute $D^{-\nu} Q_{in}$ using (39) and save the result in Q_{out}.
delev	Q_{in}	Q_{out}	Returns the Trajectory1D object Q_{out} of the same shape as Q_{in} of degree $p_{out} = p_{in} + 1$. Knots and coefficients are computed with (13) and (15), respectively.
dredact	Q_{in}	Q_{out}	Returns, if possible, the Trajectory1D object Q_{out} of the same shape as Q_{in} of degree $p_{out} = p_{in} - 1$. Knots and coefficients are computed with algorithm proposed in [23].
sum	$Q_{in\,1}, Q_{in\,2}$	Q_{out}	Returns the sum of the Trajectory1D objects $Q_{in,\,1}$ and $Q_{in,\,2}$ with the same degree $p = p_1 = p_2$ and the same knot domain ($t_{s,\,1} = t_{s,\,2}$, $t_{e,\,1} = t_{e,\,2}$), first using a knot refinement algorithm [18] to make the curves compatible, then using (42).
mult	$Q_{in\,1}, Q_{in\,2}$	B_{out}	Returns a new UbsplObject instance B_{out} with degree $p_{out} = p_{in\,1} + p_{in\,2}$ computed by Piegl and Tiller algorithm [27].

Algorithm 1 and Algorithm 2 are just two examples of Macros, even quite simple. In general terms, a sufficiently extensive library of Macro-instructions allows the designer a rapid prototyping of trajectories through an I/O Block programming approach, where each block could be a Macro or a function of the framework. Under this point of view, we could think of a framework implementation that includes a visual programming environment, or incorporating the proposed method into one that is already commercially available (e.g. Simulink® [32]).

3.2. Mechanical cam profile benchmark

In order to evaluate our approach in terms of stability and robustness, this section proposes a benchmark taken from [8] in the field of mechanical cam profile design, and compares the results obtained with IPTP and the CamOMiLe language [8,9]. As is customary in this application field, the angular velocity of the cam ω is assumed to be constant and the "geometric" quantities of the follower are adopted. It is called geometric velocity $y' = \dot{y}/\omega = dy/d\theta$ and geometric acceleration $y'' = \ddot{y}/\omega^2 = d^2y/d\theta^2$ [33].

Algorithm 3 SetValueSm Macro-instruction

Input:

1. Trajectory1D objects $Q_{in\,1}$, $Q_{in\,2}$.
2. Point1D object P.
3. Total lift h.
4. Periods T_1 and T_2 according to Fig. 5.
5. Tolerance TOL.
6. Incremental step ε and step ratio ϕ.

Output:

1. Trajectory1D object Q_{out}.

Algorithm:

$\Delta \leftarrow (P.ord - Q_{in\,1}.ord)$
Compute $D^{\Delta}Q_{in\,1}$ using (36) or (39)
Evaluate $D^{\Delta}Q_{in\,1}$ in $P.t$ and save the value in $\bar{h}$
$P.val \leftarrow (P.val - \bar{h})$
$Q_{in\,2} \leftarrow$ SetValueAm ($Q_{in\,2}$, P) ▷ Algorithm 1
Compute the sum between $D^{\Delta}Q_{in\,1}$ and $Q_{in\,2}$ and save the result in Q_{out}
Compute $D^{-\Delta}Q_{out}$ using (36) or (39)
$Q_{out} \leftarrow D^{-\Delta}Q_{out}$
return Q_{out}

In general, for the ith derivative holds $d^i y/d\theta^i = d^i y/(\omega^i \cdot dt^i)$. Consider the following double-dwell cam specification:

- $J^{peak} = 0.045$ mm/deg^3.
- 1th rise of 40 mm in 30 deg.
- 2th rise of 30 mm in 40 deg. End the rise at zero velocity.
- High dwell at 70 mm for 30 deg.
- Fall of 70 mm in 80 deg by seven-phases trajectory with cycloidal blend. The length of each phase must be: $\Delta t_0 = \Delta t_1 = \Delta t_2 = \Delta t_4 = \Delta t_5 = \Delta t_6 = 10$ deg, $\Delta t_3 = 20$ deg.

The proposed solution is presented in schematic form in Fig. 6 as a block diagram.

First, Algorithm 2 is performed using a cosine impulse J_{in} of amplitude J^{peak} and width A, defined by solving system (61) under condition (64) by adopting an equispaced grid of 11 points. The remaining input parameters are set as follows: $h = 40$ mm, $T_1 = 15$ deg, $T_2 = 15$ deg, $TOL = 10^{-5}$mm, $\varepsilon = 0.1$, $\phi = 0.1$. The resulting trajectory piece defines the first rise piece, which is used as input into the JerkLimRise macro-instruction (algorithm 2) to obtain the complete rise. With reference to Fig. 7, this Macro requires a jerk-limited displacement ramp J_{in} with blend width T_1, linear phase width T_2 and total lift h_1. First, a new piece of trajectory is created from J_{in} to define a negative displacement ramp (dashed green line in Fig. 7), with blend width T_1 and linear phase width $T_3 - T_1$. The blend phase of the new piece is obtained by changing sign to the blend phase of J_{in}, and scaling it vertically by an α scale factor. In the limit case of $\alpha = 1$, the zero-velocity constraint at the end of the motion is imposed, but the total lift achieved generally deviates from the desired ones. Assuming that the desired lift is greater than that achieved in this limit case (condition $h_2^f > h_2^i$ in Fig. 7), we can impose the constraint by defining an iterative loop that starts from the condition of $\alpha = 1$, and redefines the trajectory by decreasing α by the parameter ε for each iteration. The zero-velocity constraint at the end of the motion is maintained during iterations using the SetValueSm Macro-instruction (Algorithm 3), which adds a new and appropriately scaled blend phase of width T_1 at the end of the trajectory piece. Note that this solution involves the condition $T_3 \geq 2 \cdot T_1$. The input parameters in JerkLimitRise are set as follows: $h = 70$ mm, $T_1 = 15$ deg, $T_2 = 15$ deg, $T_3 = 40$ deg, $TOL = 10^{-5}$mm, $\varepsilon = 0.01$, $\phi = 0.1$. In the jerk derivation order, the high dwell is

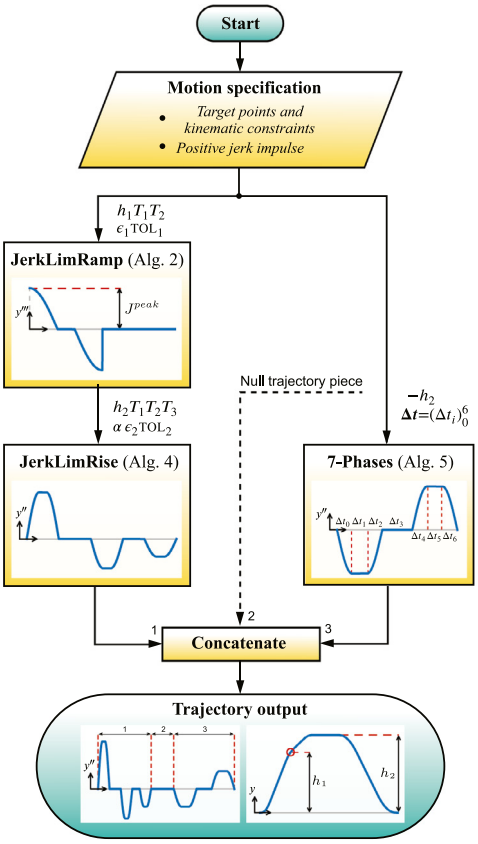


Fig. 6. Block diagram of the cam profile benchmark solution.

represented by a simple null piece of width 30 deg. The fall phase of $h = -70$ mm is instead entirely defined by the SevenPhases Macro-instruction (Algorithm 5), which generalizes a typical 7-phase constant acceleration trajectory [34] to any type of blend. Algorithm 5 also requires a positive jerk pulse J_{in} as input, and the same one already used for Algorithm 2 is adopted. Finally, rise, dwell and fall pieces, are concatenated using the same pattern shown in Fig. 1 in order to define the output trajectory. Note that the solution of Fig. 6 is parameterized on the positive impulse J_{in}, which can be defined of any type as long as the condition $2 \cdot A \leq T_1$ of Algorithm 2 is satisfied.

Fig. 8(a) shows the SVAJ resulting from the benchmark compared with the results of the CamOMiLe software language. For more information, we report the proof of the differentiation method implemented in the CamOMiLe compiler in the Appendix. To summarize, the main advantages of IPTP are:

- *Robustness.* IPTP is not affected by numerical problems typical of methods processing streams of points, e.g., differentiation instability (mainly localized at the ends of the trajectory and in discontinuities), drift phenomena (also described in [35, 36]), and undesired smoothing during repeated integration and differentiation operations.
  Fig. 8(b) shows integration drift in the high dwell phase due to composite trapezoidal rule processing. In this case, IPTP limits drift phenomena only to the component due to finite-precision arithmetic, which usually do not affect the functionality of the application.
  Fig. 8(c) shows the spikes in the discontinuities caused by numerical differentiation. IPTP avoids these instabilities by modeling discontinuities and sharp corners with multiple knots and coefficients.

Algorithm 4 JerkLimRise Macro-instruction

Input:

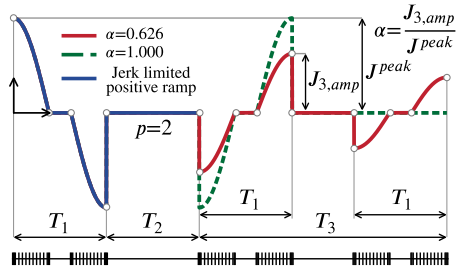
1. Trajectory1D object J_{in}.
2. Total lift h.
3. Periods T_1, T_2 and T_3 according to Fig. 7(a).
4. Tolerance TOL.
5. Incremental step ε and step ratio ϕ.

Output:

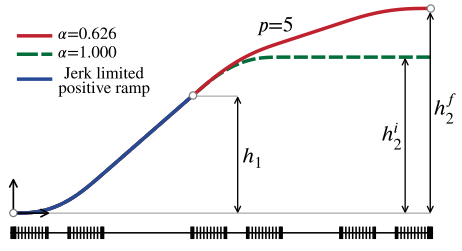
1. Trajectory1D object J_{out}.

Algorithm:

Initialize two pieces of null trajectory, J_2 and J_4, with (62) on domain $[0..1]$
Initialize an empty Point1D object P
$P.val \leftarrow 0.0$; $P.ord \leftarrow 1$; $P.t \leftarrow \sum_{i=1}^{3} \Delta T_i$
$\alpha \leftarrow 1$ ▷ initialize α scale factor
repeat
  repeat
    $\alpha \leftarrow \alpha - \varepsilon$
    Split J_{in} into T_1 with the pattern shown in Fig. 1 and save the left piece in J_3
    Vertically scale J_3 of $K = -\alpha$ using (71)
    Concatenate J_{in}, J_3 with the pattern shown in Fig. 1 and save the result in J_1
    Adapt J_2 in the new domain $[0..T_3 - T_1]$ with (69)
    Adapt J_4 in the new domain $[0..T_3 + T_2]$ with (69)
    Concatenate J_1, J_2 with the pattern shown in Fig. 1 and save the result in J_1
    Concatenate J_4, J_3 with the pattern shown in Fig. 1 and save the result in J_3
    Compute $D^{-2}J_3$ using (39) and save the result in V_3
    $V \leftarrow$ SetValueSm (J_1, V_3, P) ▷ Algorithm 3
    Compute $D^{-1}V$ using (39)
    Evaluate $D^{-1}V$ in $P.t$ and save the value in $\bar{h}$
  until $(\bar{h} < h)$
  $\alpha \leftarrow \alpha + \varepsilon$; $\varepsilon \leftarrow \varepsilon \cdot \phi$
until $(\|\bar{h} - h\| > TOL)$
Compute D^2V using (36) and save the result in J_{out}
return J_{out}



(a) Jerk.



(b) Displacement.

Fig. 7. Schematic representation of JerkLimRise Macro-instruction (Algorithm 4).

- *Numerical consistency.* CamOMiLe limits the minimum design tolerance due to the propagation of numerical errors that may be close to the design tolerance and do not lead the method to converge. For example, in our benchmark the minimum tolerance that we were able to set in order to ensure a stable solution using CamOMiLe is $TOL = 10^{-2}$. IPTP limits the design tolerance only to finite-precision arithmetic.
- *High computational performance.* In general, IPTP provides better computational performance than CamOMiLe. The main reason lies in the redundant interpolations used by the CamOMiLe compiler. These operations, in the language called "filter" functions, are necessary to make compatible two data sets defined on non-coherent grids. IPTP does not perform these operations at all in exact solutions, while it performs only one interpolation in approximate solutions (when the trajectory piece is created).
  In terms of memory consumption, IPTP adopts a more compact format than CamOMiLe and typically leads to smaller file sizes. Table 2 reports a performance analysis of this benchmark for different types of J_{in} impulses, performed with library PAPI (Performance Application Programming Interface [37]) with an Intel® Core™ i3-3217U CPU on Ubuntu 20.04.3 LTS operating system. For each test, the number of floating point operations (FLOP) required by the algorithm and the number of double-precision floating point data consumed to store the output trajectory are reported.

It is brought to the reader's attention that these remarks are plausibly extendable to other numerical solutions that process streams of points.

Algorithm 5 SevenPhases Macro-instruction

Input:

1. Trajectory1D object J_{in}.
2. Total lift h.
3. Periods vector $\boldsymbol{\Delta t} = \{\Delta t_0, \ldots, \Delta t_6\}$.

Output:

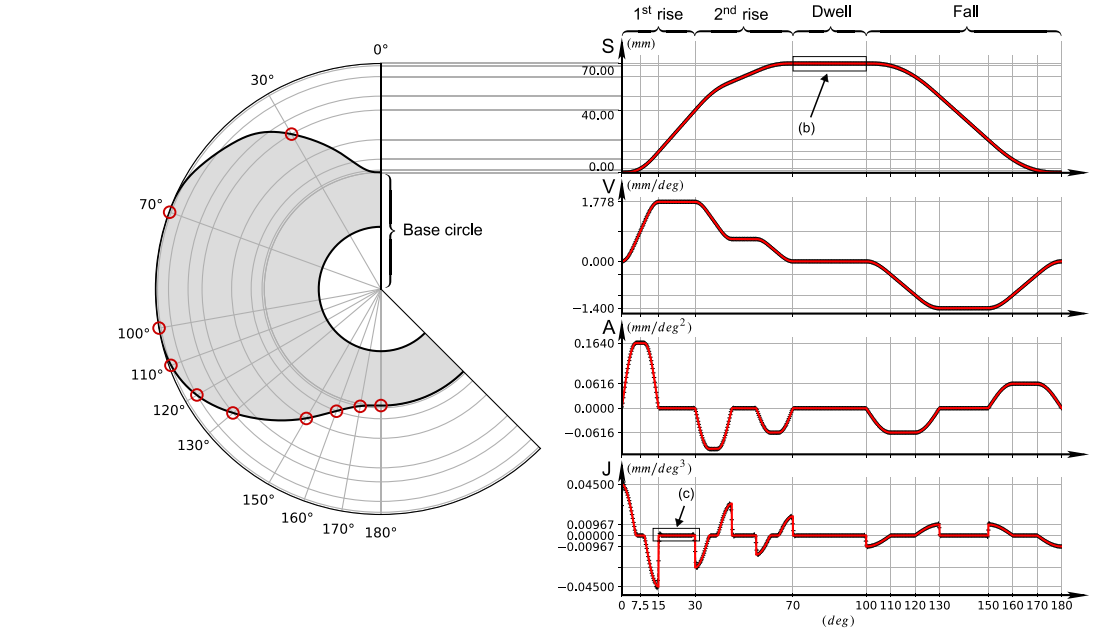
1. Trajectory1D object J_{out}.

Algorithm:

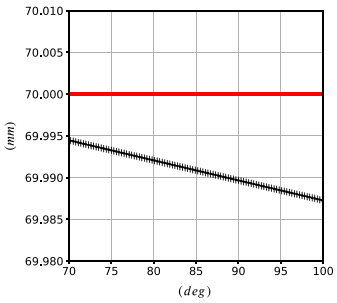
Adapt J_{in} in the new domain $[0..\Delta t_0]$ with (69) and save the result in J_0
Initialize a null piece J_1 with (62) on domain $[0..\Delta t_1]$
Mirror J_0 using (66) and save in the new instance J_2
Vertically scale J_2 of $K = -1$ using (71)
Adapt J_2 in the new domain $[0..\Delta t_2]$ with (69)
Initialize a null piece J_3 with (62) on domain $[0..\sum_{i=3}^{6} \Delta t_i]$
Concatenate J_0, J_1, J_2, J_3 with the pattern shown in Fig. 1 and save the result in J_l
Adapt J_{in} in the new domain $[0..\Delta t_4]$ with (69) and save the result in J_4
Initialize a null piece J_5 with (62) on domain $[0..\Delta t_5]$
Mirror J_4 using (66) and save in the new instance J_6
Vertically scale J_6 of $K = -1$ using (71)
Initialize a null piece J_3 with (62) on domain $[0..\sum_{i=0}^{3} \Delta t_i]$
Concatenate J_3, J_4, J_5, J_6 with the pattern shown in Fig. 1 and save the result in J_r
Compute $D^{-2}J_r$ using (39) and save the result in V_r
Initialize an empty Point1D object P
$P.val \leftarrow 0.0$; $P.ord \leftarrow 1$; $P.t \leftarrow \sum_{i=0}^{6} \Delta t_i$
$J_{out} \leftarrow J_l \leftarrow$ SetValueAm (V_r, P) ▷ Algorithm 1
return J_{out}

4. Case study

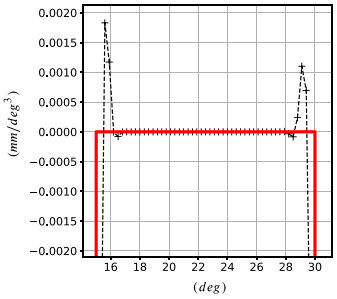
This section presents a further example that goes in the direction of greater generality and flexibility of use of the IPTP framework. Of course, this example too does not exhaust the



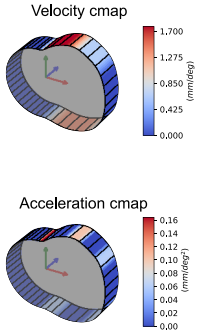
(a) SVAJ.

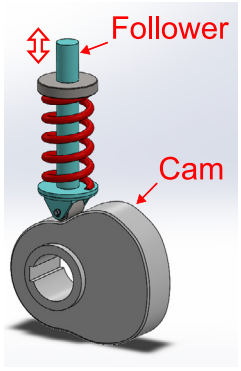


(b) Integration drift.



(c) Spikes in discontinuities.





(d) 3D model.

Fig. 8. Mechanical cam profile benchmark results. IPTP solution in solid red line and CamOMiLe solution in dashed black line with plus markers.

possible ways to use it, but it is intended to indicate a possible direction for the development of more complex and bespoken solutions. In particular, we show how, by defining four fundamental types of trajectory pieces, it is possible to build up a law of motion that satisfies an arbitrary number of kinematic constraints by combining a certain number of these fundamental elements with a building block approach. It should be noted that some of the kinematic constraints to be satisfied can express physical characteristics of the machine to be programmed, such as the maximum velocity or maximum acceleration that can be reached by a joint.

Here, the algorithm developed with our open-source library, the hardware configuration and the results are presented.

4.1. TOTP algorithm with jerk limits

Let $\boldsymbol{p} = (p_i)_{i=0}^n$ be a set of kinematic targets $p_i = \{h_i, v_i, a_i\}$, with h_i, v_i, a_i the ith position, velocity and acceleration, respectively, and $\boldsymbol{\tau} = (\tau_i)_{i=0}^n$ the vector of unknown time abscissas associated with $\boldsymbol{p}$. Let $\boldsymbol{r}(\tau)$ be the B-spline representing the displacement trajectory, the problem is mathematically formulated as: *Constraints:*

$$\boldsymbol{r}(\tau_i) = h_i, \quad \text{for } i = 0, \dots, n \tag{72}$$

Table 2
Mechanical cam profile benchmark performance evaluation.

Method	Unitary impulse type[a]	# of Para.	MFLOP[b]
IPTP			
(TOL $= 10^{-5}$)	$\cos(x)\|_{[0,\pi/2]}$, $p = 3$, $n = 11$	383	314
	$(1 - x^2)\|_{[0,1]}$	113	98
	$(1 - x^3)\|_{[0,1]}$	153	136
	1	53	28
CamOMiLe			
(TOL $= 10^{-2}$)	$\cos(x)\|_{[0,\pi/2]}$, $\Delta\theta = 0.3$ deg	604	3207

[a]The impulse J_{in} is obtained by multiplying by the amplitude J^{peak}. Additional parameters are: $p :=$ degree of the B-spline; $n :=$ number of interpolation points; $\Delta\theta :=$ sampling step.

[b]Computations performed with the library PAPI (Performance Application Programming Interface [37]) with an Intel® Core™ i3-3217U CPU. The analysis evaluates the PAPI_DP_OPS counter on Ivy Bridge architecture.

$$D^1\boldsymbol{r}(\tau_i) = v_i, \quad \text{for } i = 0, \dots, n \tag{73}$$

$$D^2\boldsymbol{r}(\tau_i) = a_i, \quad \text{for } i = 0, \dots, n \tag{74}$$

$$\|D^3\boldsymbol{r}(\tau)\|_\infty \leq J^{peak}, \quad \forall\tau \in [\tau_0, \tau_n] \tag{75}$$

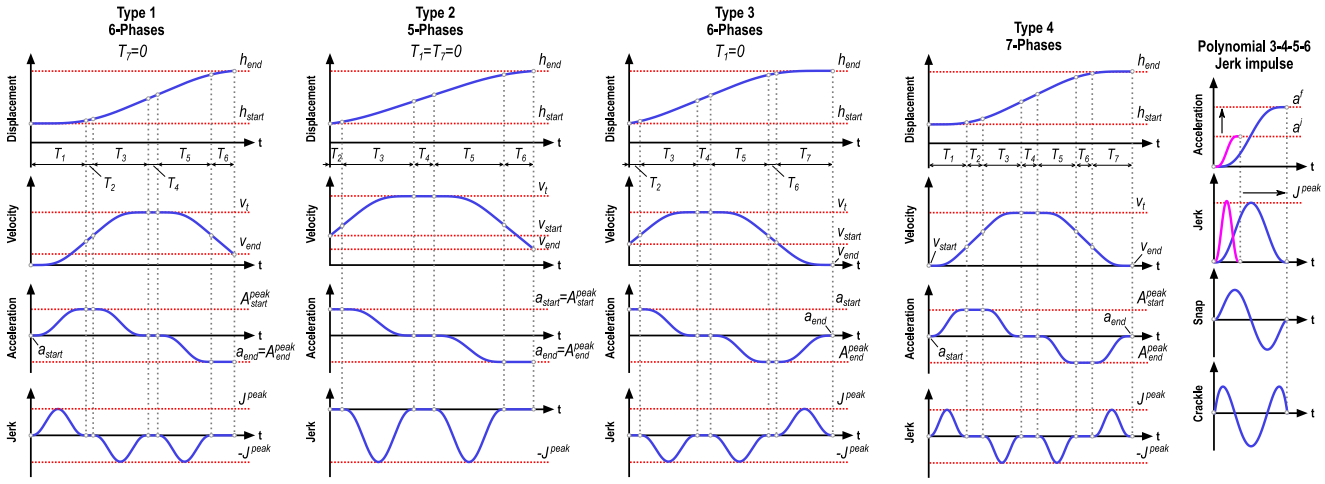


Fig. 9. Four types of motion profiles with constrained kinematics up to jerk and continuous up to crackle. On the right, the polynomial 3-4-5-6 jerk impulse used in the motion profiles.

Objective function:

$$\text{minimize } T = \sum_{i=0}^{n-1}(\tau_{i+1} - \tau_i) = \sum_{i=0}^{n-1} \Delta\tau_i \tag{76}$$

We also define the vector $\boldsymbol{v_t} = (v_{t,i})_{i=0}^{n-1}$, with $v_{t,i}$ the travel velocity set between the target p_i and p_{i+1}. Note that $\|v_{t,i}\|$ is generally not the maximum velocity for $\tau \in [\tau_i, \tau_{i+1}]$, but it could be between v_i and v_{i+1}. To ensure smoothness, acceleration and deceleration ramps defined by the parameters A^{peak}_{start} and A^{peak}_{end} are also added at the beginning and at the end of the trajectory.

With reference to Fig. 9, the algorithm we propose is divided into two cases:

1. $\#\boldsymbol{p} = 6$ (only two targets). A single 7-phase piece is used (Type 4 in Fig. 9). This algorithm is significantly different from Algorithm 5 where the periods vector $\boldsymbol{\Delta t}$ is set. In this case the algorithm computes the periods vector in order to minimize the objective function (76).
2. $\#\boldsymbol{p} > 6$. The first and the last piece use 6-phase trajectory algorithms (Type 1 and Type 3 in Fig. 9, respectively). In the central pieces a 5-phase trajectory (Type 2 in Fig. 9) is used.

The four types of motion shown in Fig. 9 are computed with the same logic. Here, we only expose Type 4 and the other types are considered to be clear. With reference to the ith piece, the constraints (72), (73) and (74) are indicated below with the subscript "start" and "end", to indicate respectively i and $i+1$. For example, h_{start} and h_{end} denote h_i and h_{i+1}, respectively.

Let J_{in} be a Trajectory1D object of a positive jerk impulse with amplitude J^{peak}. The times T_1, T_3, T_5, T_7 are computed by imposing the acceleration constraints Δa_i to J_{in}, with $i \in \{1, 3, 5, 7\}$, using the Macro-instruction SetValueEn (Algorithm 6). These constraints are defined as:

$$\Delta a_1 = A^{peak}_{start} - a_{start} \tag{77}$$
$$\Delta a_3 = -A^{peak}_{start} \tag{78}$$
$$\Delta a_5 = A^{peak}_{end} \tag{79}$$
$$\Delta a_7 = -(A^{peak}_{end} - a_{end}) \tag{80}$$

Algorithm 6 imposes the constraint by increasing the impulse width while maintaining its amplitude J^{peak}, according to (75) and (76). The time T_i also depends on the type of impulse J_{in} inserted in Algorithm 6, getting the minimum value for a constant jerk impulse. The resulting pieces J_1, J_3, J_5, J_7 are then corrected in sign multiplying each coefficient by $\Delta a_i / \|\Delta a_i\|$, with $i \in \{1, 3, 5, 7\}$.

Algorithm 6 SetValueEn Macro-instruction

Input:

1. Trajectory1D object Q_{in}.
2. Point1D object P.

Output:

1. Trajectory1D object Q_{out}.

Algorithm:

1: $t_s \leftarrow Q_{in}.\boldsymbol{t}[0]$; $t_e \leftarrow Q_{in}.\boldsymbol{t}[-1]$
2: $\Delta \leftarrow (P.ord - Q_{in}.ord)$
3: **assert** $\Delta > 0$ ▷ invalid for non-positive values
4: Compute $D^{-\Delta}Q_{in}$ using (39)
5: Evaluate $D^{-\Delta}Q_{in}$ in t_e and save the value in $\bar{h}$
6: **assert** $\bar{h} \neq 0$ ▷ division by zero error
7: $k \leftarrow \sqrt[\Delta]{\|P.val/\bar{h}\|}$
8: Adapt Q_{in} in the new domain $[t_s..(kt_e - (k-1)t_s)]$ with (69) and save the result in Q_{out}
9: **return** Q_{out}

Let $\boldsymbol{j}_i(\tau)$ be the B-spline relative to the jerk piece J_i, the times T_2 and T_6 are computed in order to set (73), according to (76), by:

$$\Delta v_1 = \int_0^{T_1} \left(\int_0^t \boldsymbol{j}_1(\tau) d\tau + a_{start} \right) dt \tag{81}$$
$$\Delta v_3 = \int_0^{T_3} \left(\int_0^t \boldsymbol{j}_3(\tau) d\tau + A^{peak}_{start} \right) dt \tag{82}$$
$$\Delta v_5 = \int_0^{T_5} \left(\int_0^t \boldsymbol{j}_5(\tau) d\tau \right) dt \tag{83}$$
$$\Delta v_7 = \int_0^{T_7} \left(\int_0^t \boldsymbol{j}_7(\tau) d\tau + A^{peak}_{end} \right) dt \tag{84}$$
$$\Delta v_2 = v_t - v_{start} - \Delta v_1 - \Delta v_3 \tag{85}$$
$$\Delta v_6 = v_{end} - v_t - \Delta v_5 - \Delta v_7 \tag{86}$$
$$T_2 = \Delta v_2 / A^{peak}_{start} \tag{87}$$
$$T_6 = \Delta v_6 / A^{peak}_{end} \tag{88}$$

Similarly, T_4 is computed by imposing (72), in accordance with (76), by:

$$\Delta h_1 = \int_0^{T_1} \left(\int_0^s \left(\int_0^t \boldsymbol{j}_1(\tau) d\tau + a_{start} \right) dt + v_{start} \right) ds \tag{89}$$
$$\Delta h_2 = T_2 \left(A^{peak}_{start} \cdot T_2/2 + v_{start} + \Delta v_1 \right) \tag{90}$$

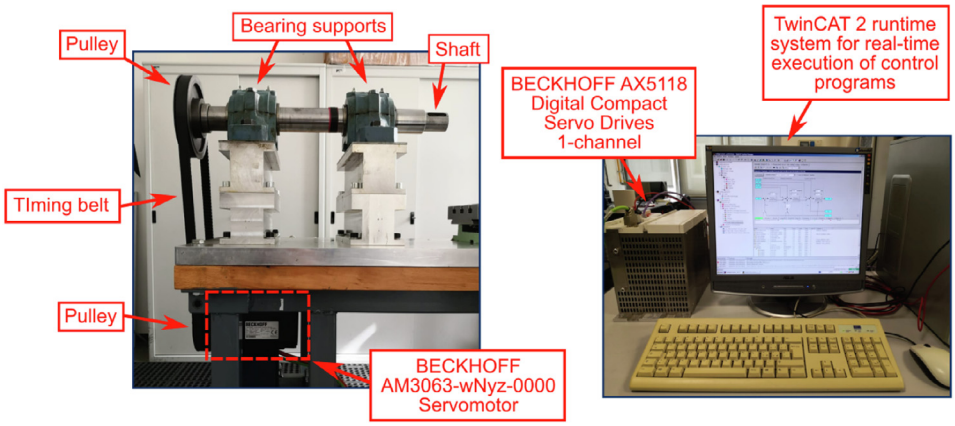


Fig. 10. Two photos of the experimental setup, showing mechanical system, servo motor (17 N m, 5.03 kW, 5000 min^{-1}), single-channel servo drives and real-time control program.

$$\Delta h_3 = \int_0^{T_3} \left(\int_0^s \left(\int_0^t \boldsymbol{j}_3(\tau) d\tau + A_{start}^{peak} \right) dt + v_{start} + \right. \\ \left. + \sum_{i=1}^{2} \Delta v_i \right) ds \tag{91}$$

$$\Delta h_5 = \int_0^{T_5} \left(\int_0^s \int_0^t \boldsymbol{j}_5(\tau) d\tau dt + v_t \right) ds \tag{92}$$

$$\Delta h_6 = T_6 \left(A_{end}^{peak} \cdot T_6/2 + v_t + \Delta v_6 \right) \tag{93}$$

$$\Delta h_7 = \int_0^{T_7} \left(\int_0^s \left(\int_0^t \boldsymbol{j}_7(\tau) d\tau + A_{end}^{peak} \right) dt + v_t + \right. \\ \left. + \sum_{i=5}^{6} \Delta v_i \right) ds \tag{94}$$

$$\Delta h_4 = h_{end} - h_{start} - \sum_{i=1}^{3} \Delta h_i - \sum_{i=5}^{7} \Delta h_i \tag{95}$$

$$T_4 = \Delta h_4 / v_t \tag{96}$$

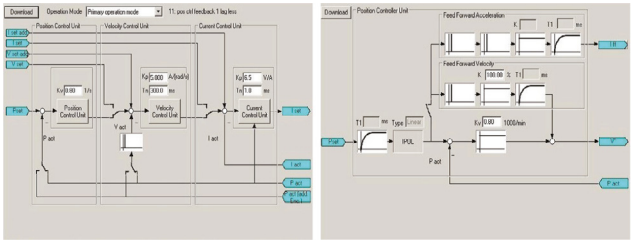
The jerk pieces J_1, J_3, J_5, J_7 are then concatenated with interposed null jerk pieces of width T_2, T_4, T_6, respectively computed with (87), (96) and (88), using the method reported in Section 2.2.3. Finally, the trajectory is integrated 3 times, and the constant a_{start}, v_{start}, h_{start} is added respectively for acceleration, velocity and position.

The remaining motion profiles (Type 1 to 3 in Fig. 9) are obtained by suppressing some phases in Type 4. For example, for Type 1, phase seven is suppressed and the constraints $T_7 = 0$ and $A_{end}^{peak} = a_{end}$ are imposed.

4.2. Experimental setup

The system used in the experimental test is shown in Fig. 10. It consists of a Beckhoff AM3063-wNyz-0000 servo motor [38] that moves a shaft of known inertia through a timing belt drive. Real-time servomotor control is performed by the Beckhoff AX5118 servo drive [38], in turn controlled with EtherCAT® I/O bus from a PC running the softPLC controller TwinCAT2, which runs in "ring 0" execution mode ("kernel" mode) with higher priority than normal Windows programs run in "ring 3" mode ("user" mode) [39]. The servo motor control logic, schematized in Fig. 11, consists of a typical cascade control closed-loop system. In our case we used a proportional position feedback control and a velocity feedforward. For further specifications on control logic, refer to the Beckhoff documentation [39].

The trajectory planned by the algorithm of Section 4.1, is set in the control logic as a position set-point by means of an electronic cam table. In this table a profile is described by a large number of position points, i.e. the positions of the master axis (virtual axis) with corresponding positions of the slave axes (real axes). With a mechanical analogy we can assimilate the angular



(a) (b)

Fig. 11. Driver control parameters in TwinCAT2 [39] environment. The two figures are connected by the light blue labels. (a) Cascade control closed-loop system: current loop inside the velocity loop which is in turn inside the position loop. (b) Position controller unit with velocity feedforward and proportional position feedback control. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

movement $\theta(t)$ of a shaft driving the cam to the positions of the master axis, while the points of the slave ones correspond to movements $P_i(\theta(t))$ of followers, numbered with $i = 1, \ldots, n$. In other words, the points in the cam table represent the equidistant interpolation points of the slave axes on the modulo range of the master axis, ranging from 0 to 2π. The position of the master axis is hence implicitly defined, and it is not shown in the aforementioned table. In the test rig here exploited, only one slave axis is available and operated, but this does not indicate any limit in applying the proposed method. Indeed, nowadays commercial axis cards allow to interpolate in synchronized motion even up to hundreds slave axes through cam tables. Therefore, a cam table has been implemented to indicate the possible use of the proposed approach with a general purpose equipment, which can be easily extended to multi-axis applications.

In this application we use a cyclic cam table, and the tested trajectory is repeated for each revolution. The timing law of the master axis $\theta(t)$ is set as a S-curve jerk-limited motion profile with maximum acceleration and jerk imposed. However, in the test the acceleration and deceleration transients are not significant, as the cam table is repeated at each revolution and the data are collected in a revolution where the speed has already reached the steady-state value. In the results reported below it is therefore correct to assume a constant master axis angular velocity (hypothesis typically adopted in the design of mechanical cam profiles). Let T be the total time of the planned trajectory in Section 4.1, the speed value set on the master axis is:

$$\omega_{master} = \dot{\theta}(t) = (2\pi)/T \tag{97}$$

Table 3
Desired positions and kinematic limits for all tasks. For all tasks: $J^{peak} = 1500\,\mathrm{rad/s^3}$.

Task	Motion type	h_{start} (rad)	h_{end} (rad)	v_{start} (rad/s)	v_{end} (rad/s)	v_{travel} (rad/s)	a_{start} (rad/s^2)	a_{end} (rad/s^2)	A^{peak} (rad/s^2)	$\Delta\tau$ (s)
1	Type 1	0.0	π	0.0	15.7	8.8	0.0	75.4	75.4	0.433
2	Type 2	π	4π	15.7	25.1	25.1	75.4	0.0	–	0.399
3	Type 2	4π	$29\pi/5$	25.1	3.1	25.1	0.0	−75.4	–	0.354
4	Type 2	$29\pi/5$	$27\pi/5$	3.1	0.0	−3.1	−75.4	37.7	–	0.533
5	Type 2	$27\pi/5$	$31\pi/5$	0.0	−6.3	7.5	37.7	−37.7	–	0.772
6	Type 2	$31\pi/5$	3π	−6.3	6.3	−12.6	−37.7	75.4	–	1.029
7	Type 2	3π	$18\pi/5$	6.3	−18.8	12.6	75.4	−100.0	–	0.565
8	Type 2	$18\pi/5$	0.0	−18.8	0.0	−28.3	−100.0	75.4	–	0.605
9	Type 2	0.0	π	0.0	9.4	6.3	75.4	37.7	–	0.522
10	Type 2	π	3π	9.4	15.7	12.6	37.7	37.7	–	0.499
11	Type 2	3π	5π	15.7	22.0	18.8	37.7	37.7	–	0.334
12	Type 2	5π	$34\pi/5$	22.0	0.0	23.1	37.7	−113.0	–	0.350
13	Type 2	$34\pi/5$	$28\pi/5$	0.0	−12.6	−12.6	−113.0	0.0	–	0.358
14	Type 2	$28\pi/5$	$24\pi/5$	−12.6	6.3	−12.6	0.0	37.7	–	0.576
15	Type 2	$24\pi/5$	$28\pi/5$	6.3	−12.6	−9.4	37.7	−37.7	–	0.964
16	Type 3	$28\pi/5$	0.0	−12.6	0.0	−18.8	−37.7	0.0	75.4	1.143

4.3. Results

In order to smooth the trajectory to the crackle,[1] a polynomial 3-4-5-6 jerk impulse is used in input to the algorithm of Section 4.1, defined as:

$$\boldsymbol{p}(\tau) = -64\tau^6 + 192\tau^5 - 192\tau^4 + 64\tau^3, \quad \forall\tau \in [0, 1] \tag{98}$$

It should be noted that this choice is aimed at highlighting the potential of the proposed method rather than at obtaining a specific practical result on the system. Indeed, the high smoothness provided by a such high degree polynomial law would require a more refined motion control and its full exploitation is beyond the scope of this work.

The kinematic constraints used in our example are summarized in Table 3. The same table also reports the times $\Delta\tau_i = \sum_{j=1}^{m} T_j$, with m the number of phases of the ith piece. The total time of the resulting trajectory is $T = 9.436$ s. Note that we have set the additional condition $p_0 \equiv p_n$ necessary to set up a cyclic cam table.

Fig. 12 shows the SVAJ of the planned trajectory, the kinematic constraints and the trajectory actually followed by the servo drive. The acceleration is compared with the torque law collected by a current sensor integrated in the servomotor, and expressed as a percentage with sign.

The proposed algorithm is tested in a controlled environment for a simple single-axis application, however it is easily generalizable in more complex contexts for synchronized multi-axis systems. In this case the targets would be provided by a motion controller in the matrix form $\boldsymbol{p} = (p_{i,j})_{i=0,j=0}^{n-1\ m-1}$, with n the number of synchronized via-points and m the number of slave axes to be synchronized, and the algorithm of Section 4.1 can be run parallel to all m columns to generate the trajectories of the servomotors.

5. Conclusions

This paper proposes a new framework for off-line trajectory planning that uses time-parameterized B-splines to represent joint trajectories. We define a piecewise trajectory by concatenating elementary pieces, and impose constraints combining closed-form techniques with a numerical approach, extending and generalizing known methodologies. Our approach allows the development of scalable and reusable applications through macro-instructions definition. IPTP offers a more stable and robust solution than similar approaches based on a sampled representation of the trajectory. Specifically, in the case of CamOMiLe language compiler [8,9], our solution allows to eliminate numerical instabilities that involve important limitations.

In future work, we plan to test this methodology in the development of a multi-objective trajectory planning algorithm for a complex multi-axis machine, combining geometric constraints to avoid collisions in the Cartesian space with reduction of residual vibrations and energy consumption.

[1] Crackle is the 5th time-derivative of the position.

CRediT authorship contribution statement

Marco Riboli: Conceptualization, Methodology, Software, Data curation, Validation, Writing – original draft, Writing – review & editing. **Fabio Corradini:** Software, Validation. **Marco Silvestri:** Conceptualization, Software, Supervision. **Alessandra Aimi:** Formal analysis, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix

Let $\boldsymbol{x} = (x_i)_{i=0}^{n}$ be the equispaced abscissas of the sempled data $\boldsymbol{f} = (f_i)_{i=0}^{n}$, with $f_i = f(x_i)$ and the constant step size $h = x_{i+1} - x_i$. Let $\boldsymbol{F} = (F_i)_{i=0}^{n}$ be a cumulative integrals vector, with:

$$F_i = \int_{x_0}^{x_i} f(x)dx \tag{99}$$

approximated in the CamOMiLe compiler with the composite trapezoidal rule:

$$F_i \simeq \sum_{j=0}^{j=i-1} (h/2)(f_j + f_{j+1}), \quad i = 1, \ldots, n \tag{100}$$

Proposition (*Silvestri [8, p. 34–45]*)**.** *Let $\boldsymbol{F}$ be a cumulative integrals vector of the unknown function $f(x)$ computed with (100). The value f_i, with $i = 2, \ldots, n-2$, can be estimated with the following 5-points formula:*

$$f_i \simeq (F_{i-2} - 6F_{i-1} + 6F_{i+1} - F_{i+2})/(8 \cdot h) \tag{101}$$

Proof. Assuming $\boldsymbol{F}$ result of (100) starting from $\boldsymbol{f}$, with reference to Fig. 13, inverting the procedure means searching for the trapezoids that generated the rectangles area $\boldsymbol{\Delta}_i$. For each trapezoid it

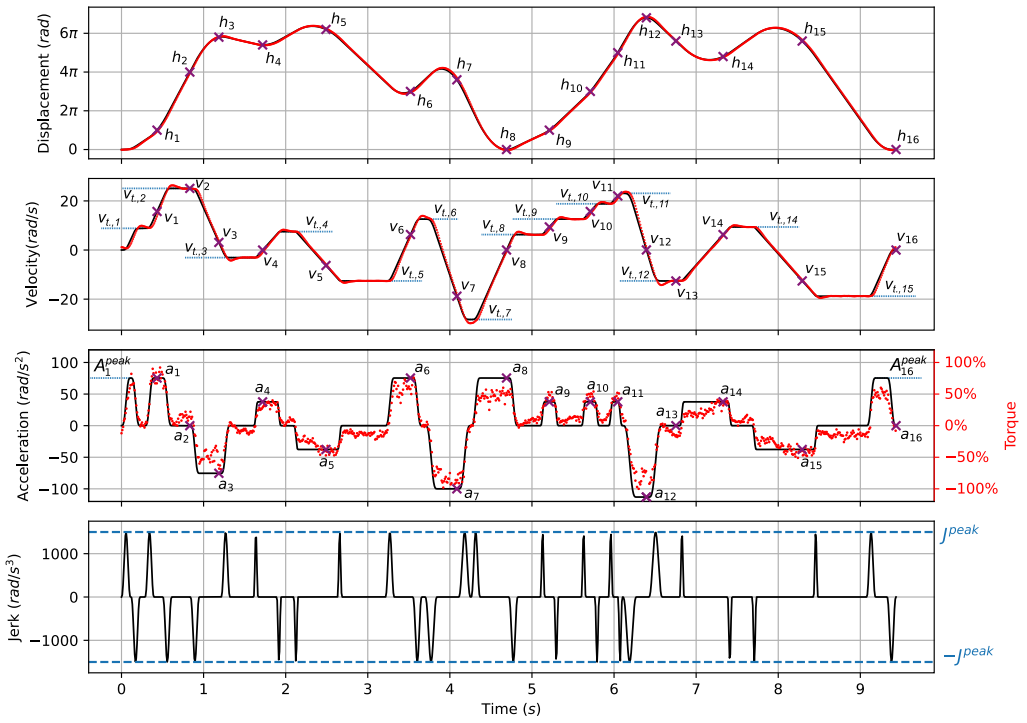


Fig. 12. Comparison between planned trajectory (black solid) and collected data (red dots). The torque supplied by the motor is reported as a percentage of the maximum torque. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

must be:

$$\mathcal{A}\left(\overline{x_i x_{i+1} f_{i+1} f_i}\right) = F_{i+1} - F_i = \boldsymbol{\Delta}_i \tag{102}$$

$$\widehat{x_i f_i f_{i-1}} \equiv \widehat{x_i f_i f_{i+1}} \tag{103}$$

$$\widehat{x_{i+1} f_{i+1} f_i} \equiv \widehat{x_{i+1} f_{i+1} f_{i+2}} \tag{104}$$

The problem is not determined, with one degree of freedom maintained. For the geometric construction that leads to the differentiation estimate, refer to Fig. 13. Let us consider the four polylines indicated by $\boldsymbol{a}_{\mathrm{V},i} = a_{\mathrm{V},i}(x)$, $\boldsymbol{b}_{\mathrm{V},i} = b_{\mathrm{V},i}(x)$, $\boldsymbol{c}_{\mathrm{V},i} = c_{\mathrm{V},i}(x)$ and $\boldsymbol{d}_{\mathrm{V},i} = d_{\mathrm{V},i}(x)$. These curves are defined with the criterion of presenting a zero slope corresponding to a specific timing span around x_i. In particular, $\boldsymbol{a}_{\mathrm{V},i}, \ldots, \boldsymbol{d}_{\mathrm{V},i}$ are constants in the spans $[x_{i-2}, x_{i-1})$ $[x_{i-1}, x_i)$, $[x_i, x_{i+1})$ and $[x_{i+1}, x_{i+2})$, assuming values Δ_{i-2}/h, Δ_{i-1}/h, Δ_i/h and Δ_{i+1}/h with $\Delta_i = F_{i+1} - F_i$ and $i = 0, \ldots, n-1$, respectively.

Having denoted by $\boldsymbol{s} = s(x)$ a generic polyline related to the knots $x_{i-2}, \ldots, x_{i+2}$, we will consider such type of functions under the following constraint:

$$\begin{aligned}\min(a_{\mathrm{V},i}(x), b_{\mathrm{V},i}(x), c_{\mathrm{V},i}(x), d_{\mathrm{V},i}(x)) \le s(x) \le \\ \le \max(a_{\mathrm{V},i}(x), b_{\mathrm{V},i}(x), c_{\mathrm{V},i}(x), d_{\mathrm{V},i}(x))\end{aligned} \tag{105}$$

$\forall x \in [x_{i-2}, x_{i+2}]$. With reference to Fig. 14, discarding all the curves $s(x)$ that do not respect (105) means excluding all solutions that have the value of the derivative of opposite sign at each span. This choice is justified for our application, where, if $s(x)$ does not verify (105), an incorrect step is probably adopted to sample the trajectory.

A polyline that satisfies (105) can be defined as

$$s_{\mathrm{V}}(x) = (a_{\mathrm{V},i}(x) + b_{\mathrm{V},i}(x) + c_{\mathrm{V},i}(x) + d_{\mathrm{V},i}(x))/4 \tag{106}$$

yielding the estimate

$$f_i \simeq s_{\mathrm{V}}(x_i) = (a_{\mathrm{V},i}(x_i) + b_{\mathrm{V},i}(x_i) + c_{\mathrm{V},i}(x_i) + d_{\mathrm{V},i}(x_i))/4 \tag{107}$$

By repeating the same procedure considering only the three points x_{i-1}, x_i, x_{i+1} and such type of functions under the constraint:

$$\begin{aligned}\min(b_{\mathrm{III},i}(x), c_{\mathrm{III},i}(x)) \le s(x) \le \\ \le \max(b_{\mathrm{III},i}(x), c_{\mathrm{III},i}(x))\end{aligned} \tag{108}$$

$\forall x \in [x_{i-1}, x_{i+1}]$, a second estimate of f_i is defined by:

$$f_i \simeq s_{\mathrm{III}}(x_i) = (b_{\mathrm{III}}(x_i) + c_{\mathrm{III}}(x_i))/2 \tag{109}$$

Hence, a further estimate of f_i is:

$$\begin{aligned}f_i \simeq &(s_{\mathrm{V}}(x_i) + s_{\mathrm{III}}(x_i))/2 = \\ =&(a_{\mathrm{V}}(x_i) + b_{\mathrm{V}}(x_i) + c_{\mathrm{V}}(x_i) + d_{\mathrm{V}}(x_i) + \\ &+ 2b_{\mathrm{III}}(x_i) + 2c_{\mathrm{III}}(x_i))/8\end{aligned} \tag{110}$$

Defining for convenience $\boldsymbol{\delta} := (\delta_i)_0^{n-1}$ with

$$\delta_i = \Delta_i/h = (F_{i+1} - F_i)/h\,, \quad i = 0, \ldots, n-1 \tag{111}$$

We have:

$$a_{\mathrm{V}}(x_i) = \delta_{i-2} + 2(\delta_{i-1} - \delta_{i-2}) = 2\delta_{i-1} - \delta_{i-2} \tag{112}$$

$$b_{\mathrm{V}}(x_i) = b_{\mathrm{III}}(x_i) = \delta_{i-1} \tag{113}$$

$$c_{\mathrm{V}}(x_i) = c_{\mathrm{III}}(x_i) = \delta_i \tag{114}$$

$$d_{\mathrm{V}}(x_i) = \delta_{i+1} + 2(-\delta_{i+1} + \delta_i) = -\delta_{i+1} + 2\delta_i \tag{115}$$

From (110) for substitution:

$$\begin{aligned}f_i \simeq &(2\delta_{i-1} - \delta_{i-2} + \delta_{i-1} + \delta_i - \delta_{i+1} + \\ &+ 2\delta_i + 2\delta_{i-1} + 2\delta_i)/8 = \\ =&(-\delta_{i-2} + 5\delta_{i-1} + 5\delta_i - \delta_{i+1})/8\end{aligned} \tag{116}$$

Substituting (111) in (116) we find:

$$f_i \simeq (F_{i-2} - 6F_{i-1} + 6F_{i+1} - F_{i+2})/(8 \cdot h) \quad \square$$

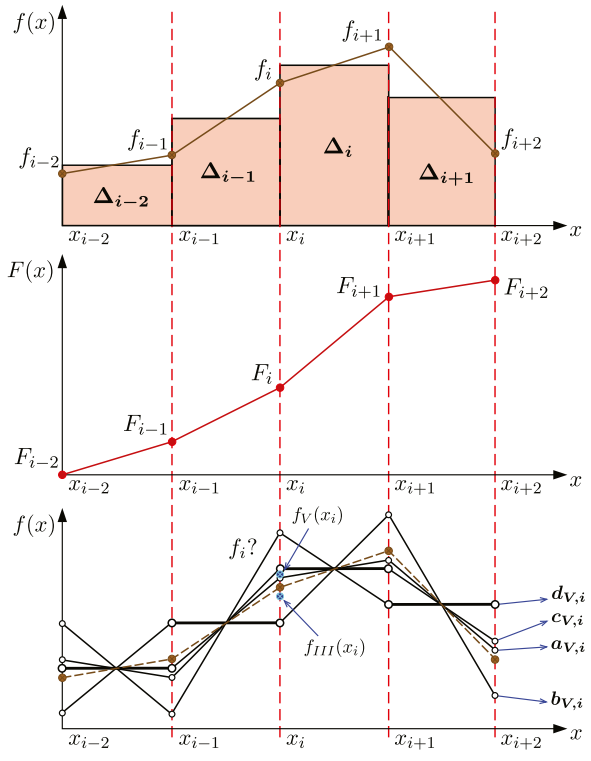


Fig. 13. Geometric construction for derivative estimation.

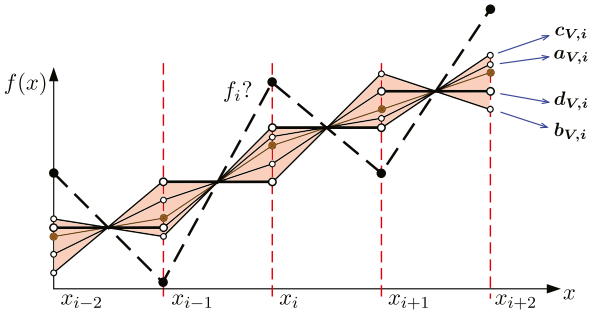


Fig. 14. Graphic representation of the discarded segments.

Remark. Eq. (101) does not provide the estimate of f_0, f_1, f_{n-1}, f_n. The CamOMiLe compiler provides the following estimates:

$$f(x_0) \simeq (14\delta_0 - 8\delta_1 + 2\delta_2)/8 \tag{117}$$

$$f(x_1) \simeq (2\delta_0 - 8\delta_1 - 2\delta_2)/8 \tag{118}$$

$$f(x_{n-1}) \simeq (2\delta_{n-1} - 8\delta_{n-2} - 2\delta_{n-3})/8 \tag{119}$$

$$f(x_n) \simeq (14\delta_{n-1} - 8\delta_{n-2} + 2\delta_{n-3})/8 \tag{120}$$

For other remarks and tests on the shape preservation properties of the method see [8,9].

References

[1] Liu Huashan, Lai Xiaobo, Wu Wenxiang. Time-optimal and jerk-continuous trajectory planning for robot manipulators with kinematic constraints. Robot Comput-Integr Manuf 2013;29(2):309–17. http://dx.doi.org/10.1016/j.rcim.2012.08.002.

[2] Gasparetto A, Zanotto V. A new method for smooth trajectory planning of robot manipulators. Mech Mach Theory 2007;42(4):455–71. http://dx.doi.org/10.1016/j.mechmachtheory.2006.04.002.

[3] Gasparetto Alessandro, Zanotto Vanni. A technique for time-jerk optimal planning of robot trajectories. Robot Comput-Integr Manuf 2008;24(3):415–26. http://dx.doi.org/10.1016/j.rcim.2007.04.001.

[4] Wang Kesheng. B-splines joint trajectory planning. Comput Ind 1988;10(2):113–22. http://dx.doi.org/10.1016/0166-3615(88)90016-4.

[5] Yang Zhengyuan, Shen Li-Yong, Yuan Chun-Ming, Gao Xiao-Shan. Curve fitting and optimal interpolation for CNC machining under confined error using quadratic B-splines. Comput Aided Des 2015;66:62–72. http://dx.doi.org/10.1016/j.cad.2015.04.010.

[6] Huang Junsen, Hu Pengfei, Wu Kaiyuan, Zeng Min. Optimal time-jerk trajectory planning for industrial robots. Mech Mach Theory 2018;121:530–44. http://dx.doi.org/10.1016/j.mechmachtheory.2017.11.006.

[7] Lin Fengming, Shen Li-Yong, Yuan Chun-Ming, Mi Zhenpeng. Certified space curve fitting and trajectory planning for CNC machining with cubic B-splines. Comput Aided Des 2019;106:13–29. http://dx.doi.org/10.1016/j.cad.2018.08.001.

[8] Silvestri M. Definizione di un linguaggio formale per la descrizione di leggi di moto: Implementazione delle procedure numeriche (M.s. thesis), Università degli Studi di Parma; 1999.

[9] Cuccio A, Garziera R, Mauro S, Silvestri M, Righettini P, Riva R. Un linguaggio generale per la descrizione di leggi di moto. In: XIV AIMETA. Como; 1999.

[10] Giannelli Carlotta, Mugnaini Duccio, Sestini Alessandra. Path planning with obstacle avoidance by G1 PH quintic splines. Comput Aided Des 2016;75–76:47–60. http://dx.doi.org/10.1016/j.cad.2016.02.004.

[11] Mirzendehdel Amir M, Behandish Morad, Nelaturi Saigopal. Topology optimization with accessibility constraint for multi-axis machining. Comput Aided Des 2020;122:102825. http://dx.doi.org/10.1016/j.cad.2020.102825.

[12] Farouki Rida T, Srinathu Jyothirmai. A real-time CNC interpolator algorithm for trimming and filling planar offset curves. Comput Aided Des 2017;86:1–11. http://dx.doi.org/10.1016/j.cad.2017.01.001.

[13] Fan Wei, Gao Xiao-Shan, Yan Wei, Yuan ChunMing. Interpolation of parametric CNC machining path under confined jounce. Int J Adv Manuf Technol 2012;62. http://dx.doi.org/10.1007/s00170-011-3842-0.

[14] Zhang Lixian, Gao Xiao-shan, Li Hongbo. Multi-period turning interpolation algorithm for high-speed machining of continuous line segments with limited acceleration, Jerk and Chord error, volume volume 3: design, materials and manufacturing, parts A, B, and C. ASME International Mechanical Engineering Congress and Exposition, 2012, p. 1553–9. http://dx.doi.org/10.1115/IMECE2012-87236.

[15] Fang Yi, Qi Jin, Hu Jie, Wang Weiming, Peng Yinghong. An approach for jerk-continuous trajectory generation of robotic manipulators with kinematical constraints. Mech Mach Theory 2020;153:103957. http://dx.doi.org/10.1016/j.mechmachtheory.2020.103957.

[16] de Boor C. A practical guide to splines. Applied mathematical sciences, New York: Springer; 2001.

[17] Piegl LA, Tiller W. the NURBS book. second ed.. New York, NY, USA: Springer-Verlag; 1996.

[18] Boehm Wolfgang, Prautzsch Hartmut. The insertion algorithm. Comput Aided Des 1985;17(2):58–9. http://dx.doi.org/10.1016/0010-4485(85)90246-5.

[19] Tiller W. Knot-removal algorithms for NURBS curves and surfaces. Comput Aided Des 1992;24(8):445–53. http://dx.doi.org/10.1016/0010-4485(92)90012-Y.

[20] Piegl Les, Tiller Wayne. Software-engineering approach to degree elevation of B-spline curves. Comput Aided Des 1994;26(1):17–28. http://dx.doi.org/10.1016/0010-4485(94)90004-3.

[21] Watkins MA, Worsey AJ. Degree reduction of Bézier curves. Comput Aided Des 1988;20(7):398–405. http://dx.doi.org/10.1016/0010-4485(88)90216-3.

[22] Eck Matthias. Degree reduction of Bézier curves. Comput Aided Geom Design 1993;10(3–4):237–51. http://dx.doi.org/10.1016/0167-8396(93)90039-6.

[23] Piegl Les, Tiller Wayne. Algorithm for degree reduction of B-spline curves. Comput Aided Des 1995;27(2):101–10. http://dx.doi.org/10.1016/0010-4485(95)92150-Q.

[24] Elber Gershon. Free Form Surface Analysis Using a Hybrid of Symbolic and numeric computation (Ph.D. thesis), Utah; 1992, p. 121, December.

[25] Piegl Les, Tiller Wayne. Symbolic operators for NURBS. Comput Aided Des 1997;29(5):361–8. http://dx.doi.org/10.1016/S0010-4485(96)00074-7.

[26] Chen Xianming, Riesenfeld RF, Cohen Elaine. An algorithm for direct multiplication of B-splines. IEEE Trans Autom Sci Eng 2009;6(3):433–42. http://dx.doi.org/10.1109/TASE.2009.2021327.

[27] Piegl LA, Tiller W. Algorithm for computing the product of two B-splines. In: Le Méhauté Alain, Rabut Christophe, Schumaker L L, editors. Curves and surfaces with applications in CAGD. 1st ed.. Neshville, TN: Vanderbilt University Press; 1997, p. 337–44.

[28] Farouki RT, Rajan VT. Algorithms for polynomials in Bernstein form. Comput Aided Geom Design 1988;5(1):1–26. http://dx.doi.org/10.1016/0167-8396(88)90016-7.
[29] The MathWorks Inc. Curve fitting toolbox user's guide R2021b. Natick, Massachusetts, United State; URL https://www.mathworks.com/products/curvefitting.html.
[30] Piegl LA, Tiller W. Curve interpolation with arbitrary end derivatives. Eng Comput 2000;16(1):73–9. http://dx.doi.org/10.1007/s003660050038.
[31] Behnel Stefan, Bradshaw Robert, Citro Craig, Dalcin Lisandro, Seljebotn Dag Sverre, Smith Kurt. Cython: The best of both worlds. Comput Sci Eng 2011;13(2):31–9.
[32] Documentation Simulink. Simulation and model-based design. 2020, URL https://www.mathworks.com/products/simulink.html.
[33] Luigi P Magnani, Guido Ruggieri. Meccanismi per macchine automatiche. UTET; 1986, p. 316.
[34] Biagiotti L, Melchiorri C. Trajectory planning for automatic machines and robots. Berlin, Heidelberg: Springer Berlin Heidelberg; 2008, p. 514. http://dx.doi.org/10.1007/978-3-540-85629-0.
[35] Siciliano Bruno, Sciavicco Lorenzo, Villani Luigi, Oriolo Giuseppe. Robotics: Modelling, planning and control. In: Michael J Grimble, Michael A Johnson, editors. IEEE robotics & automation magazine. Advanced textbooks in control and signal processing, vol. 16, London: Springer London; 2009, p. 632. http://dx.doi.org/10.1007/978-1-84628-642-1.
[36] Gori I, Pattacini U, Nori F, Metta G, Sandini G. Dforc: A real-time method for reaching, tracking and obstacle avoidance in humanoid robots. In: 2012 12th IEEE-RAS international conference on humanoid robots (Humanoids 2012). IEEE; 2012, p. 544–51. http://dx.doi.org/10.1109/HUMANOIDS.2012.6651573, November.
[37] Terpstra Dan, Jagode Heike, You Haihang, Dongarra Jack. Collecting performance data with PAPI-C. In: Müller Matthias S, Resch Michael M, Schulz Alexander, Nagel Wolfgang E, editors. Tools for high performance computing 2009. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010, p. 157–73.
[38] BECHHOFF new automation technology. https://www.beckhoff.com/it-ch/products/motion.
[39] Beckhoff information system. https://infosys.beckhoff.com/.