A practical algorithm for smooth interpolation between different angular positions

A practical algorithm for smooth interpolation between different angular positions

Giovanni Legnani[b,c,a], Irene Fassi[c], Alessandro Tasora[d], Dario Fusai[d]

[a] *corresponding author, giovanni.legnani@unibs.it*
[b] *DIMI-UniBS Dipartimento Ingegneria Meccanica e Industriale, Università di Brescia, Via Branze 38, 25123 Brescia, Italy*
[c] *STIIMA-CNR Istituto di Sistemi e Tecnologie Industriali per il Manifatturiero Avanzato, National Research Council, Via Alfonso Corti 12, 20133 Milano, Italy*
[d] *Dipartimento di Ingegneria e Architettura, Università di Parma, Parco Area delle Scienze, 181/A - 43124 PARMA, Italy*

Abstract

This paper proposes a new methodology for the interpolation of a given set of 3D rotation poses that have to be reached in successive times by preserving continuity in orientation, angular velocity and angular acceleration. The discussed algorithm ensures the generation of smooth angular trajectories without singularities. The distinctive features of the proposed approach are the straightforward formulation, the reduced computational burden and the lack of iterative procedures. The presented methodology has applications in the generation of spatial motion of mechanical systems (e.g. robotics, flying devices) or in 3D computer graphics. After a theoretical introduction, the proposed algorithm is compared with other methods available in literature and some possible applications are presented.

Keywords: Smooth rotation interpolation, Angular trajectory, Rotations, Quaternions

1. Introduction

In many theoretical and application fields there is the need to interpolate between a given set of configurations in 3D space, in order to obtain a smooth motion of a rigid frame. This is a typical problem of industrial robotics (for manipulation, welding, or inspection) [16] [17], aerospace (for path planning) [29] and computer graphics areas (for camera motions) [24], but it also has applications in the field of computational solid mechanics (see, for example, [22], [7] and [3]). It is known that the motion of a rigid body in 3D space is representable by the combination of a translation and a rotation relative to a reference frame: motion interpolation is therefore the act of producing two continuous curves (one in $\mathbb{R}^3$ and the other in $\mathsf{SO}(3)$) from a set of *key frames*. The problem of spatial interpolation in Euclidean $\mathbb{R}^3$ space has been thoroughly studied and many operative algorithms are available, spacing from simple linear interpolation to smooth Bézier, B-Spline and NURBS curves. On the other hand, the interpolation of orientations still poses some difficulties, due to the peculiar characteristics of rotations in 3D space.

As a matter of fact, the orientation of solid objects is mathematically representable in many ways: rotation matrices, axis-angle formalism, Rodrigues vectors and quaternions are only part of a numerous list of possibilities. Each of these representation offers both advantages and drawbacks, making them individually suitable for a specific application and less for another. For example, if we parametrize a rotation matrix through Cardan angles (both in the intrinsic Euler $Z - X^{'} - Z^{''}$ form or in the extrinsic Tait-Bryan $X - Y - Z$ one), we obtain a straightforward and intuitive method that allows us to combine successive rotations with simple linear algebra operations (see Appendix A for more information). However, a major problem of this approach is that any adopted set of angles will have at least one configuration leading to singularity, making such choice unfit for generic applications.

A better strategy is to represent rotations by the means of unit quaternions [8]. These mathematical objects possess four components and are usually expressed as hyper-complex numbers in the form $\boldsymbol{q} = q_0 + q_1 i + q_2 j + q_3 k$ (refer to Appendix B for a deeper description). Although being less intuitive and still affected by some drawbacks (e.g. the same orientation may be expressed by two different and opposite quaternions), this formulation provides consistent advantages in the field of rotation interpolation. In fact, in order to represent a valid rotation, it is sufficient for a quaternion to have a unit norm, thus lying on the surface of a unitary 3-sphere (hypersphere). On the

other hand, a rotation matrix must be orthogonal with determinant equal to +1: from a computational point of view, the normalization of an array is much faster than the one of a 3x3 matrix (with Gram-Schmidt or similar algorithms). In addition, quaternions do not suffer of singularity problems and they only deal with four numbers instead of nine, saving valuable computer memory.

It is therefore apparent that the quaternion representation of orientations is preferable for the production of rotation curves. However, a mere linear interpolation of quaternion components is not suitable for the purpose, since it leads to non-valid rotations by cutting through the hypersphere. To overcome this problem, a certain number of proper quaternion interpolation schemes have been presented through the years. A first method for quaternion interpolation, known as SLERP (Spherical Linear intERPolation), has been put forward by Ken Shoemake in 1985 for computer animation purposes [24]: instead of a linear interpolation (LERP) between two consecutive orientations, the algorithm produces a segment of geodesic on the surface of the quaternion unitary sphere. In this way, one obtains the shortest angular path between two poses performed at constant velocity [5]. This method works nicely between two orientations, but the mere replication on multiple frames produces a segmented curve, characterized by abrupt velocity spikes at junction points. Therefore, many other efforts were made in order to improve the smoothness of the curve. In the same paper [24] Shoemake proposed to generate the equivalent of a cubic Bézier curve in S^3 space, achieving C^1 continuity. In another work the same author presented an alternative approach, which has become very popular in computer animation and is commonly referred to as SQUAD (Spherical QUADrangle interpolation) [25]. This is basically a cubic interpolation method which uses a nested SLERP algorithm between four control points; although its C^1 continuity and differentiability where not properly demonstrated in the original source [5], new proofs were given in following works (such as in [5] and [11]). Schlag [23] extended the de Casteljau construction scheme of Bézier curves to obtain B-Spline quaternion curves but not did not achieve C^2 continuity in $\mathsf{SO}(3)$. On the contrary, Pletinckx [18] produced an iterative algorithm which converged to an infinite-degree curve that, although being extremely smooth, did not have closed form formulation and required many in-between points [9]). Barr [1] presented a method to obtain Splines (simplifying a previous algorithm proposed by Gabriel et Kajiya [6]) that minimize a measure of the acceleration on curved spaces. However, this approach requires an optimization algorithm

to perform the highly computational expensive minimization. In addition, the method formulates the derivatives as finite differences resulting in a discrete sequence of points which required an additional SLERP in-between. We can mention that later Ramamoorthi [20] improved the Barr algorithm [1] by producing a faster cubic quaternion B-Spline algorithm through a new optimization criterion based on Euler-Lagrange functional. Ravani [21] investigated the possibility to extend the concept of Bézier curve to Riemannian manifolds and particularly to Lie groups. He obtained a generalized de Casteljau construction scheme which, however, still suffers of high computational burden: it is required to recursively solve multiple nonlinear differential equations and a two-point boundary value problem. Kim et al. focused on a method to generate quaternion cumulative basis B-Splines in $\mathsf{SO}(3)$ which preserved C^k continuity [9] [10]. This method produces very smooth curves but needs an iterative procedure to approximate the control points; in addition, the convergence is not assured when two consecutive orientations have large angular or axis displacements. More recently, many other techniques were investigated to tackle the problem. For example, Basarab [2] introduced a new class of quaternion interpolation curves reformulating the Kim et al. cumulative basis approach through the use of atomic functions: the curves obtained are infinitely differentiable and their curvature can be regulated by an appropriate choice of the atomic function parameters. Bolotnikov [4] provided a procedure to perform quaternion interpolation by solving the polynomial Lagrange interpolation problem from a mathematical perspective. Tan et al. [26] proposed a class of parametric quintic polynomials for quaternion interpolation, looking for a smooth, locally modifiable and efficiently-computed spline curve; the result obtained is second-order continuous and the adjustment of some tension parameters ensure the passage through given keyframes and local shape controllability. Lastly, Pu et al. [19] proposed a new type of B-Spline based on logarithmic quaternions; the problem is resolved in $\mathbb{R}^3$ and then remapped into S^3.

The method proposed in this paper aims at a different goal. We provide a simple yet effective algorithm to perform a smooth and exact interpolation of an arbitrary set of rotations without the need for iterative optimization procedures or complex mathematical formulation: thus, we address those applications where a simple implementation and a small computational overhead are high priorities. In fact, we propose a new practical interpolation scheme which preserves the desirable characteristics of SLERP (easy algorithm and constant-velocity output trajectory) while overcoming its lack of C^1 conti-

nuity at curve joints. The target is achieved with the repeated combination of SLERP interpolation segments, smoothly connected by fillet tracts generated by simple polynomial motion laws. The duration of these fillets and the angular velocity profiles can be easily and arbitrarily modified by the user to obtain the desired trajectory. With respect of other mentioned interpolation algorithms (namely quaternion B-Splines and SQUAD), the formulation is therefore easier and the computational burden is lower.

The paper is organized as follows. Section 2 formulates the algorithm, starting from the approximate interpolation of three poses (i.e. only passing through the extremes), then exposing a method to exactly interpolate all the three given poses and finally extending the exact method to any number of given key frames orientations. Section 3 presents some numerical examples of the method, comparing it with the some of the principal interpolation schemes presented in literature. Finally, examples in the field of 3D animation and in the simulation of industrial robots are also provided.

2. Interpolation between angular poses

The description of the algorithm is based on the notation and the basic properties of rotations recalled in Appendix A [12, 13]. Rotation matrix R_{ij} denotes the angular position of frame j with respect to i. Subscript (0) will be used to denote the absolute reference frame, while digits 1, 2, ... will be used to denote further reference frames. For example $\boldsymbol{u}_{(i)}$ is the representation in frame i of the unit vector $\boldsymbol{u}$.

2.1. Interpolation between 2 poses

An angular trajectory that modifies an initial orientation R_{01} into a final orientation R_{02} in "smooth" way can be obtained by rotating around a suitable unit vector $\boldsymbol{u}$ of the suitable angular value $\overline{\varphi}$ (see Appendix A). We can represent graphically this concept in Fig. 1. By operating in the absolute frame (0) we can write

$$\overline{R}(\boldsymbol{u}_{(0)}, \overline{\varphi}) R_{01} = R_{02} \qquad \overline{R}(\boldsymbol{u}_{(0)}, \overline{\varphi}) = R_{02} R_{10} \tag{1}$$

being R_{ji} the inverse of R_{ij}.
While operating in the local frame (1) from Eq. A.11 we get

$$R_{01}\overline{R}(\boldsymbol{u}_{(1)},\overline{\varphi}) = R_{02} \qquad \overline{R}(\boldsymbol{u}_{(1)},\overline{\varphi}) = R_{12} = R_{10}R_{02} \tag{2}$$

the rotation axis $\boldsymbol{u}$ and angle $\overline{\varphi}$ can be extracted by the rotation matrix using eq. A.12.
A generic intermediate orientation R between the two poses can be obtained by one of the two equivalent equations

$$\begin{aligned} R &= R_{01}\overline{R}(\boldsymbol{u}_{(1)},\varphi) \\ R &= \overline{R}(\boldsymbol{u}_{(0)},\varphi)R_{01} \end{aligned} \qquad \varphi \in [0,\overline{\varphi}] \tag{3}$$

If the rotation is applied at constant velocity for a time T, the angular velocity vector $\boldsymbol{\omega}$ expressed in frame (0) and (1) will be

$$\boldsymbol{\omega}_{(0)} = \boldsymbol{u}_{(0)}\frac{\overline{\varphi}}{T} \qquad \boldsymbol{\omega}_{(1)} = \boldsymbol{u}_{(1)}\frac{\overline{\varphi}}{T} \tag{4}$$

If the angle φ changes with a specific law of motion assuming suitable velocity, Eq. (4) will define the average velocity $\|\boldsymbol{\omega}\|$ whose unit vector will be $\boldsymbol{u}$ in any case.

2.2. Interpolation between 3 poses

If we want to rotate frame (1) into frame (3) passing through frame (2), it is possible to adopt twice the strategy of section 2.1 (see Fig. 2). Since in general the unit vectors $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ are different from each other, to preserve the continuity of angular velocity it is necessary to smoothly stop in (2) and then to start again. In the next sections we will present two methodologies to generate the desired motion without the necessity to stop but preserving the velocity and acceleration continuity.

2.3. Approximate method

In this section we present a methodology to generate a trajectory from frame (1) to frame (3) passing near frame (2) (Fig. 3). This is an approximate solution to our problem.

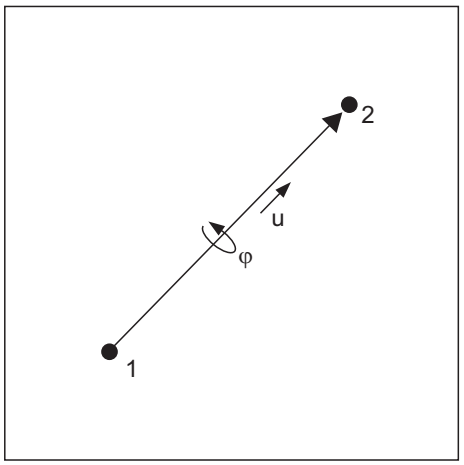


Figure 1: Conventional representation of an angular motion between two frames, consisting in a rotation of angle φ about a fixed rotation axis $\boldsymbol{u}$. The strategy adopted is described in Section 2.1.

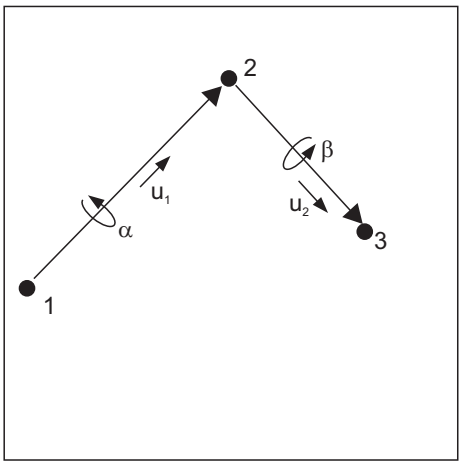


Figure 2: Conventional representation of an angular motion between three frames, starting from frame (1) and ending to frame (3) while passing through frame (2). The strategy adopted is described in Section 2.2.

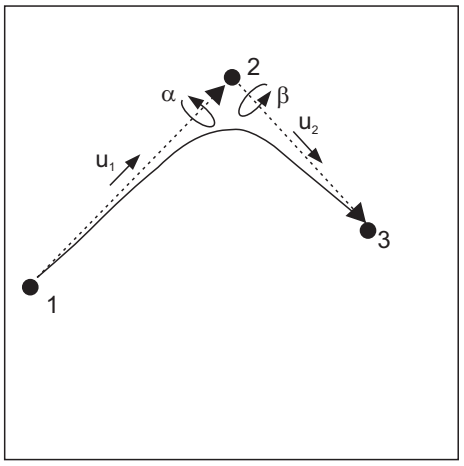


Figure 3: Approximate angular trajectory between three frames (first definition). The strategy adopted is described in Section 2.3.

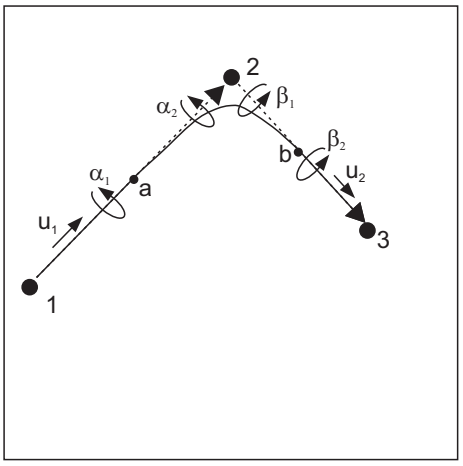


Figure 4: Approximate angular trajectory between three frames (variant to pass closer to intermediate frame). The strategy adopted is described in Section 2.3. See also Fig. 13

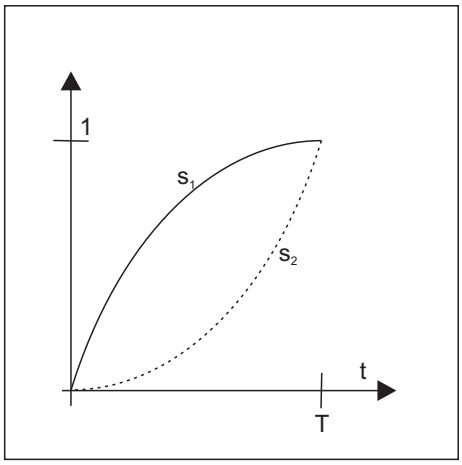


Figure 5: Unitary motion laws for the angles α and β, to be used in the approximate connection of the three frames with Eq. 7.

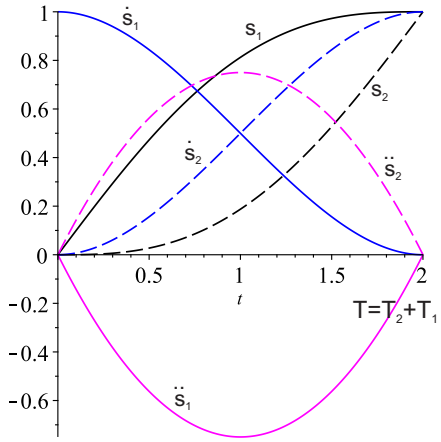


Figure 6: Position, velocity ed acceleration of the motion laws s_1 and s_2 presented in Fig. 5.

We initially observe that the frames (2) and (3) can be reached from the previous frames by suitable rotations $\overline{\alpha}$ and $\overline{\beta}$ around suitable axes $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$

$$\begin{aligned} R_{02} &= R_{01}R_{12} \qquad & R_{12} &= \overline{R}(\boldsymbol{u}_1, \overline{\alpha}) \\ R_{03} &= R_{02}R_{23} \qquad & R_{23} &= \overline{R}(\boldsymbol{u}_2, \overline{\beta}) \end{aligned} \tag{5}$$

So a trajectory from pose (1) to pose (3) can be obtained by the following equation

$$R = R_{01}\overline{R}(\boldsymbol{u}_{1(1)}, \alpha)\overline{R}(\boldsymbol{u}_{2(2)}, \beta) \qquad \alpha \in [0, \overline{\alpha}], \beta \in \left[0, \overline{\beta}\right] \tag{6}$$

where for $\alpha = \beta = 0$ we get $R = R_{01}$, while for $\alpha = \overline{\alpha}$ and $\beta = \overline{\beta}$ we get $R = R_{03}$.

To pass near frame (2) it is necessary to assign suitable initial and final conditions to the variation law of α and β. Initially it is necessary that α varies quickly and slowly in the final part. Similarly initially β should vary slowly and quickly in the final part. As an example consider the following law (Fig. 5 and Fig. 6)

$$\begin{aligned} &\alpha(t) = \overline{\alpha}s_1(t) \qquad \beta(t) = \overline{\beta}s_2(t) \qquad t \in [0, T] \\ &\dot{s}_1(0) > 0 \qquad \dot{s}_1(T) = 0 \qquad \ddot{s}_1(0) = \ddot{s}_1(T) = 0 \\ &\dot{s}_2(0) = 0 \qquad \dot{s}_2(T) > 0 \qquad \ddot{s}_2(0) = \ddot{s}_2(T) = 0 \end{aligned} \tag{7}$$

To pass closer to pose (2) it is possible to adopt the following strategy (Figure 4)

- perform an initial rotation α_1 around a $\boldsymbol{u}_1$ and reach frame (a),
- then move from frame (a) to (b) using the previously described algorithm shown in Fig. 3 and Eq. 6 with rotations α_2 and β_1
- finally move from frame (b) to (3) rotating around $\boldsymbol{u}_2$ by an angle β_2.

The mentioned operations are performed assuming $\overline{\alpha} = \alpha_1 + \alpha_2$, $\overline{\beta} = \beta_1 + \beta_2$. So the trajectory is generated by the following three steps

$$\begin{array}{lll} step\ 1: & R = R_{01}\overline{R}(\boldsymbol{u}_1, \alpha) & \alpha \in [0, \alpha_1] \\ step\ 2: & R = R_{01}\overline{R}(\boldsymbol{u}_1, \alpha_1)\overline{R}(\boldsymbol{u}_1, \alpha)\overline{R}(\boldsymbol{u}_2, \beta) & \alpha \in [0, \alpha_2], \beta \in [0, \beta_1] \\ step\ 3: & R = R_{01}\overline{R}(\boldsymbol{u}_1, \overline{\alpha})\overline{R}(\boldsymbol{u}_2, \beta_1)\overline{R}(\boldsymbol{u}_2, \beta) & \beta \in [0, \beta_2] \end{array} \tag{8}$$

In practice steps 1 and 3 are classical SLERP interpolations segments smoothly connected by step 2. The adoption of the law of motions similar to those of Figure 6 guarantees continuity of velocity and acceleration in the connection points (a) and (b) if the following conditions are assigned

$$\begin{array}{ll} s_1(0) = 0 & s_2(0) = 0 \\ \dot{s}_1(0) = \dot{\overline{\alpha}} & \dot{s}_2(0) = 0 \\ \ddot{s}_1(0) = 0 & \ddot{s}_2(0) = 0 \\ s_1(T) = \alpha_2 & s_2(T) = \beta_1 \\ \dot{s}_1(T) = 0 & \dot{s}_2(T) = \dot{\overline{\beta}} \\ \ddot{s}_1(T) = 0 & \ddot{s}_2(T) = 0 \end{array} \tag{9}$$

where $\dot{\overline{\alpha}}$ and $\dot{\overline{\beta}}$ are, respectively, the angular velocity during the first and third step of the motion. These conditions can be easily obtained by the polynomial law described in section Appendix D. By assuming $s_1 + s_2 = kt$ and so $s_2 = kt - s_1$, $\dot{s}_2 = k - \dot{s}_1$, $\ddot{s}_2 = -\ddot{s}_1$ for the particular law adopted $s_2(t) = s_1(T - t)$. Moreover $s_1(0) = 0$, $\dot{s}_1(0) = 1$, $\ddot{s}_1(0) = 0$, $s_1(T) = 1$, $\dot{s}_1(T) = 0$, $\ddot{s}_1(T) = 0$ (see Appendix D).

A numerical example of the exposed procedure is presented in Fig. 7 and Fig. 8. The law of motion can be adjusted to obtain constant angular velocity as shown in Fig. 9 and Fig. 10.

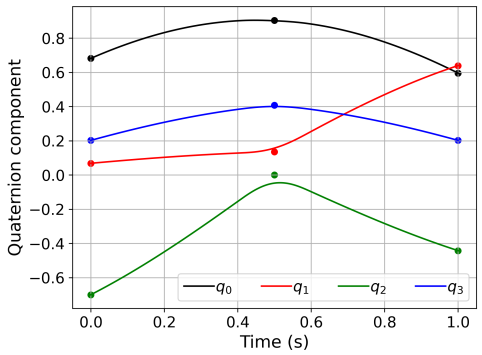


Figure 7: Numerical example of a trajectory connecting two poses and passing near to an intermediate pose (quaternion components versus time). The strategy adopted is described in 2.3.

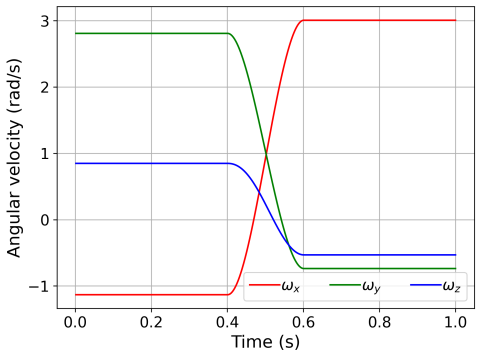


Figure 8: Angular velocity of interpolation method shown in Fig. 7.

To generate the motion with constant velocity it is possible to adopt the following numerical approach. The motion description of Figure 7 is sampled at predefined time intervals $i = 0, 1, 2, \ldots n$ then, for any time interval, the absolute value of the angular rotation is evaluated. Finally the absolute time t_i for the $i-th$ sampled orientation is determined as

$$\begin{aligned} t_0 &= 0 \\ t_i &= t_{i-1} + \frac{\Delta\varphi_i}{\|\boldsymbol{\omega}\|} \end{aligned} \tag{10}$$

where $\Delta\varphi_i$ is the angular rotation in the $i-th$ interval (between the time instant i and $i-1$) and $\|\boldsymbol{\omega}\|$ is the desired angular velocity ($\|\boldsymbol{\omega}\|$=1 for the example of Fig. 9 and Fig. 10). The proposed methodology can be simply adapted to any velocity profile by assigning the required value of $\|\boldsymbol{\omega}\|$ as a function of time. For example it is possible to start the motion with null velocity, increase it until a predefined value is reached and slowing down to smoothly stop at the end of the motion.

2.4. Exact method

In order to generate a trajectory that rotates from frame (1) to (3) and passes exactly through frame (2), it is possible to apply twice the algorithm seen in Section 2.3.

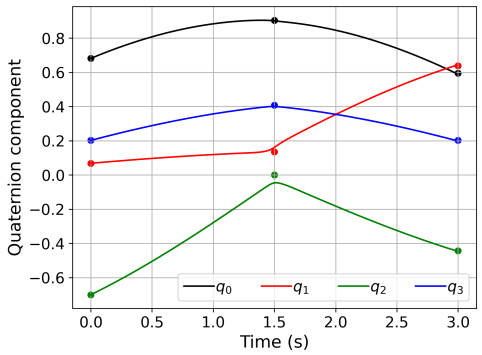


Figure 9: Numerical example of a trajectory connecting two poses and passing near to an intermediate pose, with constant angular velocity (quaternion components versus time). The strategy adopted is described in 2.3.

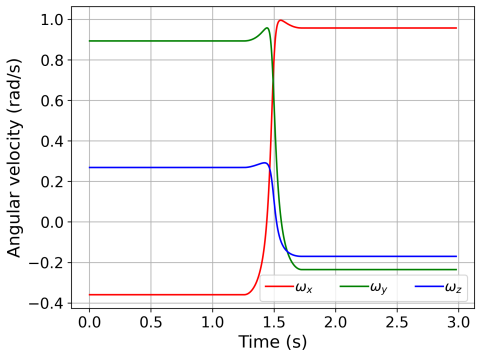


Figure 10: Angular velocity of interpolation method shown in Fig. 9.

At first (see Fig. 11), a suitable rotation axis $\boldsymbol{u}_r$ is chosen to be between $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ by using positive arbitrary weights k_1 and k_2

$$\boldsymbol{u}_r = \frac{k_1\boldsymbol{u}_1 + k_2\boldsymbol{u}_2}{\|k_1\boldsymbol{u}_1 + k_2\boldsymbol{u}_2\|} \tag{11}$$

and, chosen suitable rotation value α' e β' it is possible to generate the frames (a') e (b') by rotation around the axis $\boldsymbol{u}_r$. The intermediate auxiliary poses are determined as

$$R_{0a'} = R(\boldsymbol{u}_r, -\alpha')R_{02} \qquad\qquad R_{0b'} = R(\boldsymbol{u}_r, \beta')R_{02} \tag{12}$$

Then it is possible to apply twice the described algorithm: the first time along the poses $(1) - (a') - (2)$ and then along $(2) - (b') - (3)$. A comparison between the approximate and the exact algorithm is presented on the unitary sphere in Fig. 13 and Fig. 14, while a numerical example is reported in Fig. 15 and Fig. 16.

2.5. Exact method for arbitrary number of rotations

The presented exact algorithm can be extended to interpolate an arbitrary number of rotations. For each intermediate frame (i), i.e. from (2) to $(N-1)$, the fillet rotation axes are computed by weighted average of the axes connecting (i) with $(i-1)$ and $(i+1)$. Next, the auxiliary poses (A), (B),

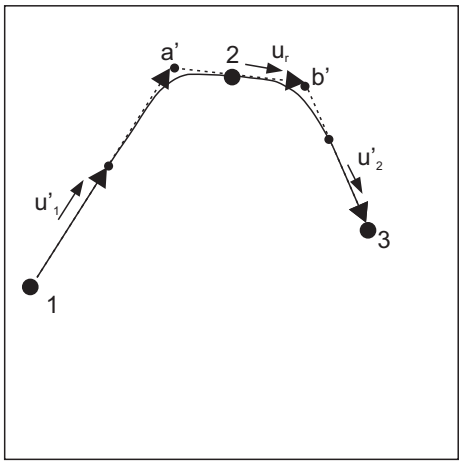


Figure 11: Exact trajectory connecting three different frames by using the intermediate poses (a') and (b'). The strategy adopted is described in 2.4.

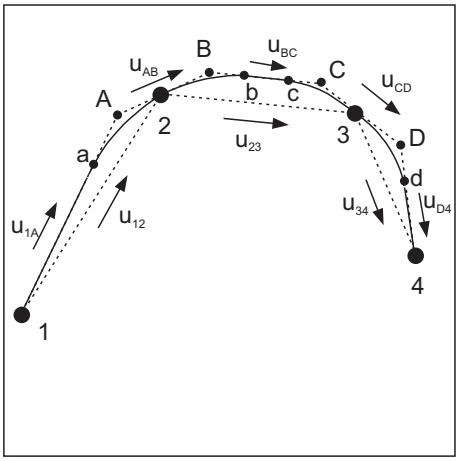


Figure 12: Exact trajectory connecting four different frames by using the intermediate poses (A), (B), (C) and (D). The strategy adopted is described in 2.5

$(C), \ldots, (X)$ (see Fig. 12) are chosen on the fillet axes by rotating of suitable desired angles. Finally, it is possible to compute the axes $\boldsymbol{u}_{1A}$, $\boldsymbol{u}_{BC}$, $\boldsymbol{u}_{D4}$, $\ldots$, $\boldsymbol{u}_{X-1,X}$ and introduce the additional knots (a), (b), (c), $\ldots$, (x) at desired angular position. For example, assuming 4 input rotations to be interpolated and referring to Fig. 12, we can create the desired trajectory through the following steps:

$$\begin{aligned}
&step\ 1: \quad R = R_{01} R(\boldsymbol{u}_{1A(1)}, \varphi_{1a}) \\
&step\ 2: \quad R = R_{0a} R(\boldsymbol{u}_{1A(1)}, \varphi_{aA}) R(\boldsymbol{u}_{AB(2)}, \varphi_{A2}) \\
&step\ 3: \quad R = R_{02} R(\boldsymbol{u}_{AB(2)}, \varphi_{2B}) R(\boldsymbol{u}_{BC(B)}, \varphi_{Bb}) \\
&step\ 4: \quad R = R_{0b} R(\boldsymbol{u}_{BC(B)}, \varphi_{bc}) \\
&step\ 5: \quad R = R_{0c} R(\boldsymbol{u}_{BC(B)}, \varphi_{cC}) R(\boldsymbol{u}_{CD(3)}, \varphi_{C3}) \\
&step\ 6: \quad R = R_{03} R(\boldsymbol{u}_{CD(3)}, \varphi_{3D}) R(\boldsymbol{u}_{D4(D)}, \varphi_{Dd}) \\
&step\ 7: \quad R = R_{0d} R(\boldsymbol{u}_{D4(D)}, \varphi_{d4})
\end{aligned} \tag{13}$$

This process can be repeated to interpolate any number of given rotations.

2.6. Assigning predefined velocity at the assigned poses

The proposed algorithm generates 3D orientation trajectories connecting the assigned poses. The angular velocity profile can be easily adjusted using the following guidelines which permit to assign exact angular velocity values

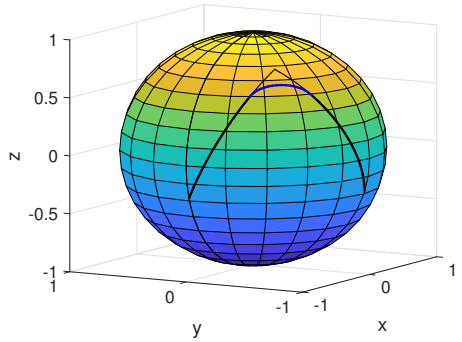


Figure 13: Representation on the unitary sphere of the approximate algorithm (see also Fig. 4).

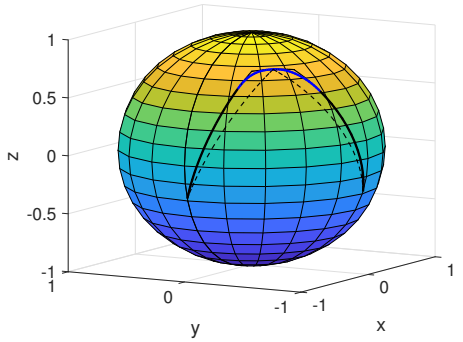


Figure 14: Representation on the unitary sphere of the exact algorithm (see also Fig. 11).

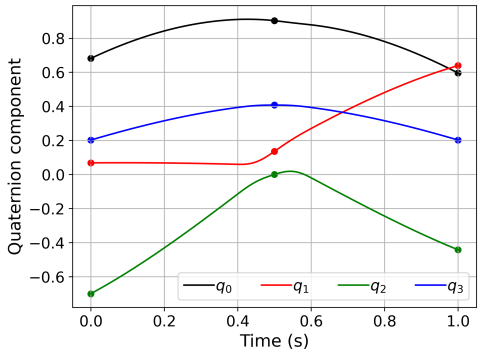


Figure 15: Numerical example of a trajectory connecting two poses and passing exactly through an intermediate pose (quaternion components versus time). The strategy adopted is described in 2.4.

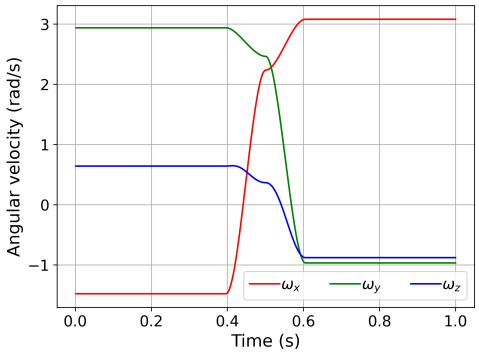


Figure 16: Angular velocity of interpolation method shown in Fig. 15.

(modulus and direction) in the initial, intermediate and final points.
To assign the direction of the angular velocity it is possible to freely assign the orientation vectors used to generate the trajectory near the points (e.g. $\boldsymbol{u}_r$ in Figure 11, and $\boldsymbol{u}_{AB}$ and $\boldsymbol{u}_{CD}$ in Figure 12), in fact the adoption of Eq. 11 is a suggestion that generally smooth the trajectory but it is not mandatory.
To assign the desired modulus of the angular velocity it sufficient to apply eq. 10 using a suitable variable value of $||\omega||$ along the trajectory. At first the whole trajectory is considered to evaluate the total rotation around the trajectory $\overline{\varphi} = \sum \varphi_i$. Then the desired law of variation of φ with respect to the time is chosen ($0 \leq \varphi \leq \overline{\varphi}$) respecting the desired velocity in the points of interest. This procedure permits to assign the desired value of $||\omega_i||$ for each time interval.

3. Numerical examples

In this section we present some numerical examples of rotation interpolation, followed by some illustrated applications.

First of all, we can quantitatively compare the presented interpolation method with other common algorithms available in literature (refer to Appendix C for further information). To this purpose, we assign some rotation key frames and observe the four quaternion component trends together with the corresponding angular velocity components, both of which with respect to time $t \in [0, 1]$. In particular, we compare the SLERP [24] method, a 2^{nd}-order quaternion B-Spline method (based on [9]), the SQUAD method (based on [25] implementation) and the proposed method. We observe that:

- SLERP passes though all the imposed frames without overshoots and minimizes the angular path. Velocity is piecewise constant but manifests abrupt discontinuities at curve joints (step variations); consequently, angular acceleration locally presents infinite values. See Fig. 17 and Fig. 18.

- 2^{nd}-order quaternion B-Spline is very smooth, but does not exactly reach the intermediate imposed frames. Velocity is continuous and mainly constituted by linear tracts, but presents drastically rapid changes at curve joints; consequently, angular acceleration will locally be very high. See Fig. 19 and Fig. 20.

- SQUAD exactly interpolates all the given poses at the cost of a slight increase of the angular path. Velocity is continuous and smooth but it manifests a continuously variable and bouncy trend. See Fig. 21 and Fig. 22.

- The proposed method produces a smooth trajectory, exactly passing through all the given poses with modest deviation from the minimal SLERP path and slight overshoots at curve joints. As for SLERP, velocity is preserved constant through the in-between interpolation tracts, but with the advantage of smooth connections at curve joints. The transitions can be arbitrarily adapted to fulfil specific local constraints. See Fig. 23 and Fig. 24.

To better visualize the achieved results in rotation interpolation, we present some snapshots of a 3D simulation as well. The task was performed by leveraging an open-source multibody simulation engine (Chrono::Engine [28]), in which the discussed algorithms were implemented.

Fig. 25 shows a free body subject to the spatial and rotational interpolation of four key poses (blue). The red/green/blue curves are integral with the X/Y/Z local axes of the moving body, while in gray we shown the stroboscopic trail of the configurations. All the poses are smoothly and exactly connected. Fig. 26 shows a comparison between the proposed method and SLERP; the four imposed rotation frames are the same as the previous ones (Fig. 25), but there is no spatial motion. One can observe the undesired spikes produced by SLERP by looking at the red/green/blue axes curves. Finally, Fig. 27 shows a potential application of the proposed algorithm: the smooth interpolation of poses performed by the end-effector of an industrial robot (namely a 6-DOF robot arm equipped with painting tools).

4. Benchmarks

In order to show that the computational effort of the proposed method is competitive with respect to other interpolation methods, we implemented a dedicated benchmark. In particular, we tested the interpolation of different number of key frames, evaluating 1E6 of intermediate samples for each. The same test has been repeated 100 times for each scenario, and the average values of computational time have been reported in Table 1. One can observe that the proposed method is indeed competitive with SLERP for a reasonably

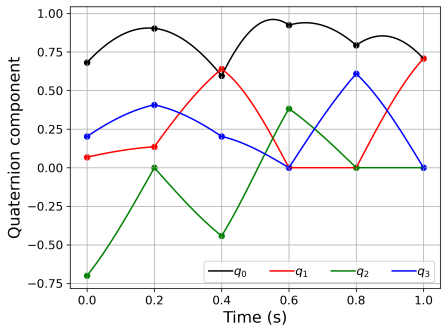


Figure 17: Numerical example of a trajectory connecting six poses using the SLERP interpolation method (quaternion components versus time). All the given frames are exactly reached by a non-smooth trajectory, having minimum angular length.

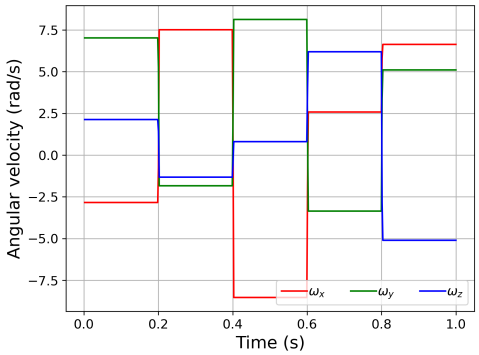


Figure 18: Angular velocity of SLERP interpolation method shown in Fig. 17. Severe discontinuities at curve joints are notable.

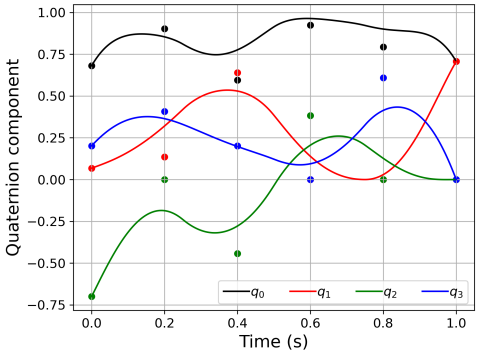


Figure 19: Numerical example of a trajectory connecting six poses using a 2^{nd}-order quaternion B-Spline (quaternion components versus time). Angular trajectory is very smooth, but does not exactly pass through all the given frames.

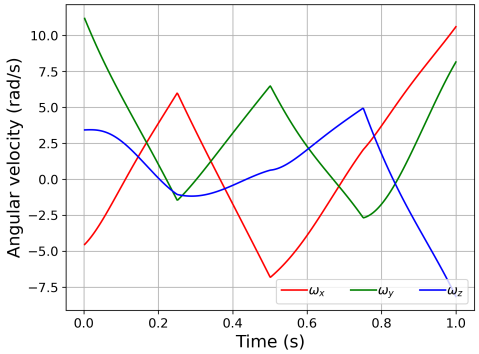


Figure 20: Angular velocity of 2^{nd}-order quaternion B-Spline interpolation method shown in Fig. 19. The trend is continuous, but with abrupt spikes at curve joints.

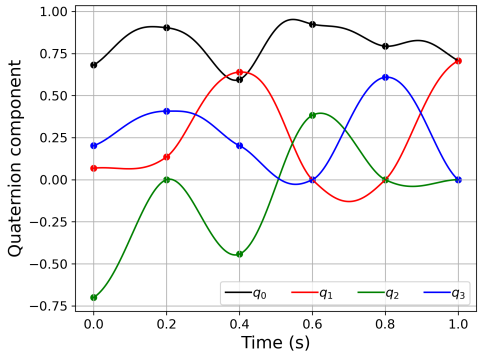


Figure 21: Numerical example of a trajectory connecting six poses using the SQUAD interpolation method (quaternion components versus time). All the given frames are exactly reached by a smooth trajectory.

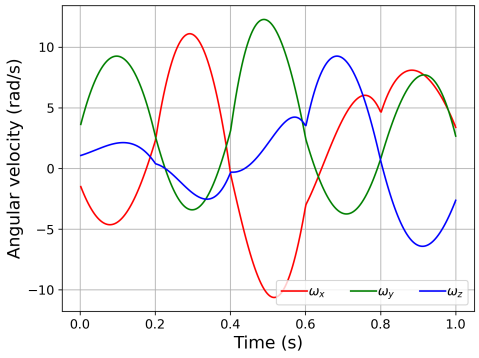


Figure 22: Angular velocity of SQUAD interpolation method shown in Fig. 21. The trend is continuous but highly oscillating.

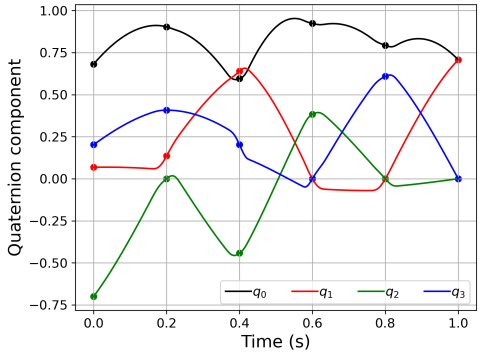


Figure 23: Numerical example of a trajectory connecting six poses using the proposed extended exact interpolation method (quaternion components versus time). All the given frames are exactly reached by a smooth trajectory.

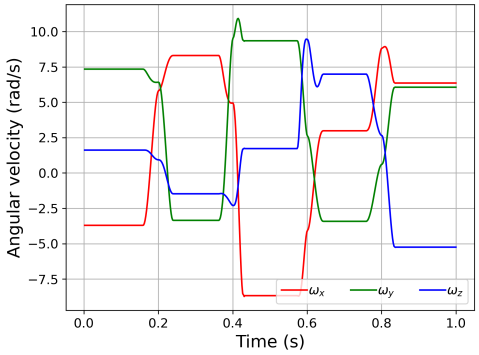


Figure 24: Angular velocity of the proposed extended exact interpolation method shown in Fig. 23. The trend remains constant over the intermediate phases and smoothly transitions from one tract to the other.

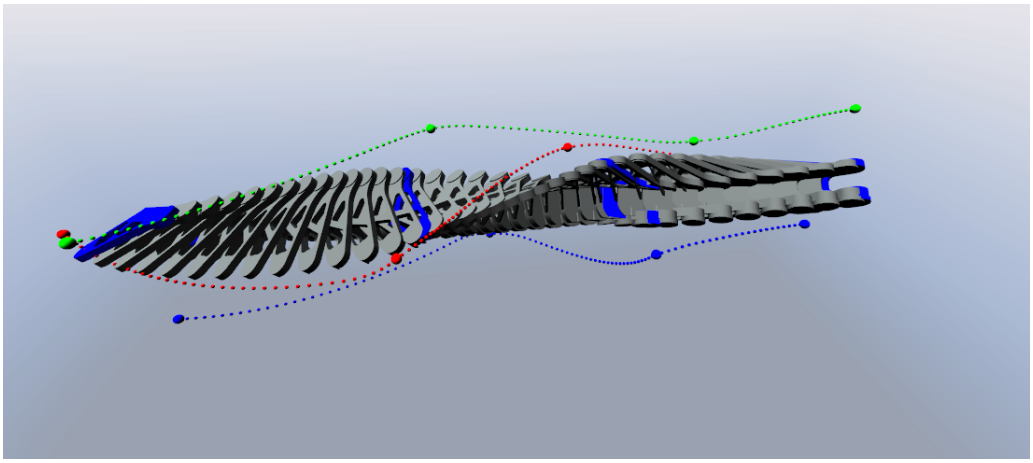


Figure 25: Simulated interpolation of four poses using the proposed method in association with a spatial motion [28]. The assigned key frames are shown in blue, while the moving body axes are represented by RGB trails (key frame passages are highlighted). It is notable that the generated trajectory is smooth and exactly passes though all the given rotations.

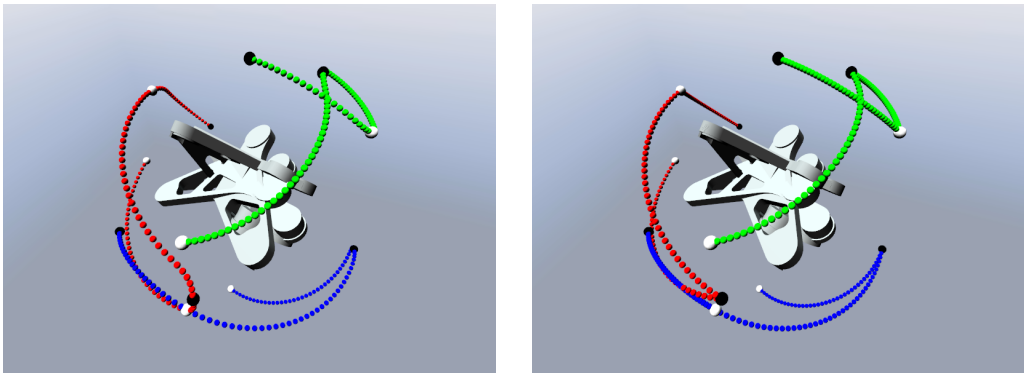
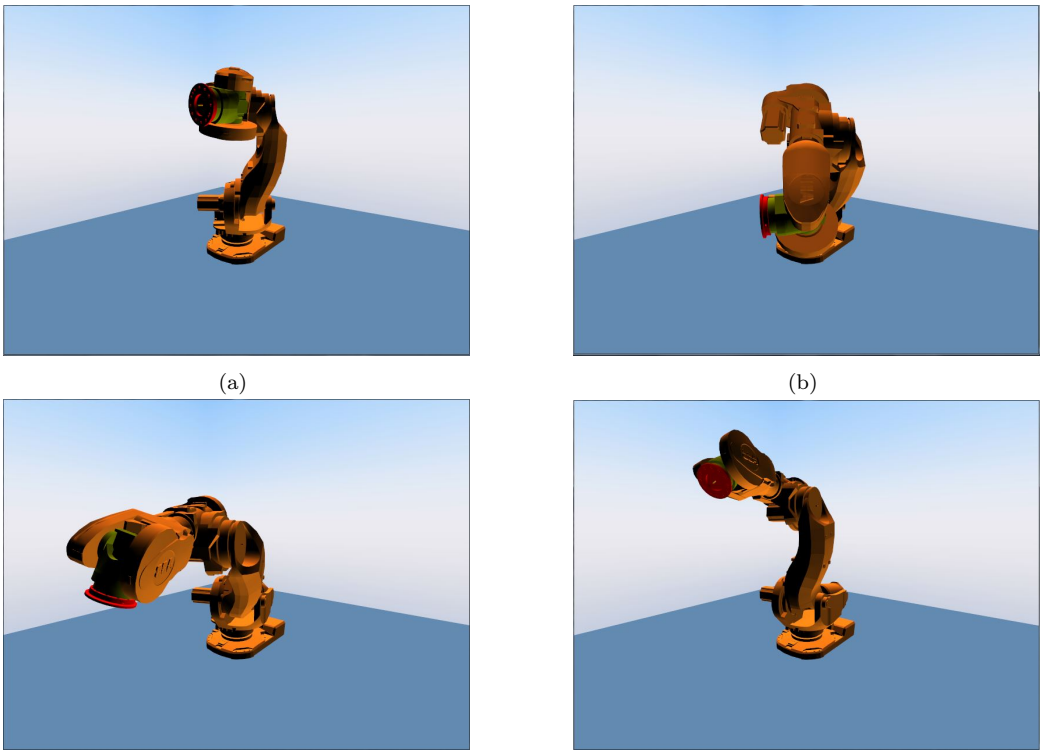


Figure 26: Simulated comparison [28] between the proposed method (left) and SLERP (right) interpolation of the same four poses of Fig.25, without spatial motion. The rotating body axes are represented by RGB trails and the passage through the assigned key frames is highlighted as white and black dots. It may be noted from the axes trails that the SLERP trajectory manifests spikes at curve joints, therefore discontinuities in angular velocity; on the contrary, the proposed algorithm connects the given orientations with smooth curves.



(a) (b) (c) (d)

Figure 27: Simulation [28] of an industrial 6-DOF robot whose end-effector, equipped with a painting tool, interpolates between various key poses using the proposed algorithm in association with a spatial motion. The provided example illustrates one of the many possible applications suitable for the algorithm.

Table 1: Computation time required for the interpolation of different numbers of given frames by the discussed methods, expressed in ms. The test involves the evaluation of 1E6 intermediate samples and has been repeated 100 times for each of them (average values reported).

	SLERP	B-Spline2	SQUAD	Proposed method
4 key frames	48.2	479.1	494.5	66.1
50 key frames	49.6	483.6	524.1	142.0
100 key frames	50.6	491.9	543.0	194.1

Table 2: Example of angular distance travelled to interpolate the same 100 random key frames by the discussed methods, expressed in rad.

	SLERP	B-Spline2	SQUAD	Proposed method
Angle	112.6	84.2	124.5	117.1

low number of given poses and, despite the progressive increase of time in function of the key frames number, its CPU time is still shorter than the 2^{nd}-order quaternion B-Spline and the SQUAD method.

The benchmarks have been computed on a Intel® Core™ i7-10510U CPU, clocked at 1.80 GHz, with 4 physical cores and 16 GB of RAM. The linear algebra formulas for the benchmark have been implemented in C++ in the Chrono::Engine open-source library [28], enabling the /O2 compiler optimization flag on the MSVC compiler, and using a single thread. In search of a fair comparison, no special optimization techniques have been put in place: in fact, the performance of the methods can be further improved using advanced programming techniques such as ASM assembly directives and AVX2 vectorization, that we have not considered at the moment.

Table 2 compares a measure of the total angular distance required by each of the methods discussed to interpolate the same 100 random key frames. It is observable that, as expected, the proposed algorithm travels a longer path with respect to SLERP but, in exchange, smoothly interpolates all the given poses. On the other hand, the angular distance travelled is slightly less if compared to SQUAD. At last, the 2^{nd}-order quaternion B-Spline presents the shortest angular path of all methods, but sacrifices the exact passage among all the given key frames.

5. Conclusions

We proposed a practical method, based on the composition of rotations and simple polynomials, that provides a smooth interpolation between an arbitrary number of given angular poses. The algorithm is rooted on repeated SLERP interpolation segments smoothly connected by suitable fillet tracts. The duration of the different segments can be arbitrarily modified by the user to optimize the trajectory. The methodology intrinsically generates transitions in orientation which preserve the continuity of angular position, velocity and acceleration. The mathematical formulation is simple because it involves simple combination of rotations and the computational burden is limited, as showed by the provided benchmark. It is easy to generate constant angular velocity trajectories as well as motions with predefined velocity profiles. Specific angular velocity vectors can be imposed in the control points. The comparison with other known methodologies shows that: 1) with respect to SLERP, it does not present velocity discontinuities but slightly increases angular distance covered and computation time; 2) with respect to a quaternion B-Spline, the computational burden is lower and it is easier to impose the exact crossing of the assigned angular positions; 3) with respect to SQUAD, the angular distance covered is smaller, the computation time is lower and there are no oscillations in velocities.

The proposed methodology can be easily used to generate 3D motions of mechanical parts and it can be useful in those cases where a simple formula for smooth interpolation of key poses is desired: we can mention, for instance, biomechanics (when one must interpolate between sample poses obtained via motion capture experiments) as well as computer graphics, video games and robotics. Finally, a possible application related to industrial robots has been presented to this end.

Appendix A. Rotation matrices

The present appendix section recalls, without proof, some fundamental notions on rotation matrices [14, 15] and defines the notation used throughout this paper.

Appendix A.1. Conversion between different reference frames

The conversion of vectors between two different reference frames (i) and (j) can be obtained by means of the rotation matrix R_{ji} representing their relative pose. Naming $\boldsymbol{v}_{(i)}$ the original three-components vector and $\underline{\boldsymbol{v}}_{(i)}$ its skew-symmetric matrix representation as in

$$\boldsymbol{v}_{(i)} = \begin{bmatrix} v_{xi} \\ v_{yi} \\ v_{zi} \end{bmatrix} \qquad \underline{\boldsymbol{v}}_{(i)} = \begin{bmatrix} 0 & -v_{zi} & v_{yi} \\ v_{zi} & 0 & -v_{xi} \\ -v_{yi} & v_{xi} & 0 \end{bmatrix} \tag{A.1}$$

we obtain the following notable relations:

$$\boldsymbol{v}_{(j)} = R_{ji}\boldsymbol{v}_{(i)} \qquad\qquad \underline{\boldsymbol{v}}_{(j)} = R_{ji}\underline{\boldsymbol{v}}_{(i)}R_{ij} \tag{A.2}$$

Each column of matrix R_{ji} contains the unit vectors of the axes of frame (i) as seen by frame (j)

$$R_{ji} = \left[\begin{array}{c|c|c} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{array}\right] = \left[\begin{array}{c|c|c} \boldsymbol{u}_x & \boldsymbol{u}_y & \boldsymbol{u}_z \\ & & \end{array}\right] \tag{A.3}$$

This matrix also represent the orientation (or attitude) of frame (i) with respect to frame (j).

Since the presented properties hold for any frames, Eq. A.2 can also be written as $\boldsymbol{v}_{(i)} = R_{ij}\boldsymbol{v}_{(j)}$; thus, we deduce that

$$R_{ij} = R_{ji}^{-1} \tag{A.4}$$

Considering that the columns of matrix R in Eq. A.3 are orthogonal to each other, we conclude that matrix R itself is orthogonal and so the following properties holds

$$R^{-1} = R^T \qquad\qquad \det(R) = \pm 1 \tag{A.5}$$

The determinant is positive if both frames are right or left, and negative if one is right and the other is left.

In case of multiple frame change it is possible to use the following composition rule

$$R_{ik} = R_{ij}R_{jk} \tag{A.6}$$

Appendix A.2. Rotation of vectors and frames

Let us consider a vector $\boldsymbol{v} = [v_x \; v_y \; v_z]^T$ to be rotated from a starting position $\boldsymbol{v}_s$ to reach a final position $\boldsymbol{v}_f$ of an angle φ around axis $\boldsymbol{u}$. This transformation can be represented by matrix R with:

$$\boldsymbol{v}_f = R\boldsymbol{v}_s \qquad R(\boldsymbol{u}, \varphi) = 1 + \underline{\boldsymbol{u}} \sin\varphi + \underline{\boldsymbol{u}}^2(1 - \cos\varphi) \tag{A.7}$$

Of course the vectors and the unit vector $\boldsymbol{u}$ must be represented by their projections in the same reference frame that we can indicate (for instance) as (r). If a rotation is expressed in frame (r), the same rotation in a different frame (s) will be (see Eq. A.2)

$$R_{(s)} = R_{sr} R_{(r)} R_{rs} \tag{A.8}$$

Let's indicate by subscript (0) the absolute reference system in which we describe the angular position of two frames (1) and (2). Remembering Eq. A.3, if we rotate the frame whose attitude is R_{01} of an angle φ around an axis $\boldsymbol{u}$ to reach the orientation R_{02}, we can write (each column of R is transformed according to Eq. A.2):

$$R_{02} = \overline{R}(\boldsymbol{u}_{(0)}, \varphi) R_{01} \tag{A.9}$$

So, if the initial angular position R_{01} and the final one R_{02} are known, the rotation which transform the former to the latter is determined in frame (0) as

$$\overline{R}(\boldsymbol{u}_{(0)}, \varphi) = R_{02} R_{01}^{-1} = R_{02} R_{10} \tag{A.10}$$

Remembering Eqs. A.2, A.6 and A.8, if the result is sought in frame (1) or (2), the rotation becomes simply

$$\overline{R}(\boldsymbol{u}_{(1)}, \varphi) = \overline{R}(\boldsymbol{u}_{(2)}, \varphi) = R_{12} \tag{A.11}$$

Once the matrix $\overline{R}$ is known, the rotation angle and axis may be extracted as

$$\begin{aligned} c &= \cos\varphi = \tfrac{x_x+y_y+z_z-1}{2} \\ s &= \sin\varphi = \tfrac{\sqrt{(y_z-z_y)^2+(x_z-z_x)^2+(x_y-y_x)^2}}{2} \\ \varphi &= \mathrm{atan2}(s,c); \qquad \pi < \varphi \leq \pi \end{aligned} \tag{A.12}$$

$$\underline{\boldsymbol{u}} = \frac{1}{2\sin\varphi}\left(\overline{R}-\overline{R}^T\right) \qquad \begin{aligned} u_x &= \tfrac{(y_z-z_y)}{2\sin\varphi} \\ u_y &= \tfrac{(z_x-x_z)}{2\sin\varphi} \\ u_z &= \tfrac{(x_y-y_x)}{2\sin\varphi} \end{aligned} \qquad \sin\varphi \neq 0$$

Appendix B. Quaternions, rotations and exponentials

We list some properties of quaternions that are used in interpolation schemes discussed in this work.

Appendix B.1. Main properties

Rotations in 3D space can be represented by four scalars q_0, q_1, q_2, q_3 called Euler parameters, related to the rotation angle φ and rotation axis $\boldsymbol{u}$ as [27]

$$\begin{gathered} q_0 = \cos\left(\tfrac{\varphi}{2}\right) \\ q_1 = u_x \sin\left(\tfrac{\varphi}{2}\right) \quad q_2 = u_y \sin\left(\tfrac{\varphi}{2}\right) \quad q_3 = u_z \sin\left(\tfrac{\varphi}{2}\right) \end{gathered} \tag{B.1}$$

By the definitions (Eq. B.1), the following constrain holds: $q_0^2+q_1^2+q_2^2+q_3^2 = 1$. Euler parameters can be interpreted as unit-length *quaternions* $\boldsymbol{q} \in \mathbb{H}_1$, $||\boldsymbol{q}|| = 1$, that is a subset of quaternion hyper-complex numbers $\mathbb{H}$ written as

$$\boldsymbol{q} = q_0 + q_1\, i + q_2\, j + q_3\, k \tag{B.2}$$

In this representation [8], the following properties hold

$$\begin{gathered} i^2 = j^2 = k^2 = -1 \\ ij = -ji = k; \qquad jk = -kj = i; \qquad ki = -ik = j \end{gathered} \tag{B.3}$$

Quaternions can be written succinctly with the notation $\boldsymbol{q} = [q_s, \boldsymbol{q}_v]$, showing the *scalar part* $q_s = q_0$ and the *vector part* $\boldsymbol{q}_v = \{q_1, q_2, q_3\}$; this

simplifies the expression of quaternion multiplication. In fact, with $\boldsymbol{\tau} \in \mathbb{H}$, $\boldsymbol{\eta} \in \mathbb{H}$, by means of the properties (B.3) one obtains the product formula:

$$\boldsymbol{\tau}\, \boldsymbol{\eta} = [\tau_s \eta s - \boldsymbol{\tau}_v \cdot \boldsymbol{\eta}_v,\ \tau_s \boldsymbol{\eta}_v + \rho_s \boldsymbol{\eta}_v + \boldsymbol{\eta}_v \times \boldsymbol{\eta}_v] \tag{B.4}$$

A rotation matrix can be constructed from a quaternion as

$$\begin{aligned} R(\boldsymbol{q}) &= \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_3 q_0) & 2(q_1 q_3 + q_2 q_0) \\ 2(q_1 q_2 + q_3 q_0) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(-q_1 q_0 + q_2 q_3) \\ 2(q_1 q_3 - q_2 q_0) & 2(q_1 q_0 + q_2 q_3) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} = \\ &= \begin{bmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{bmatrix} \end{aligned} \tag{B.5}$$

Vice versa, if the matrix R is known, the elements of the quaternion are extracted as

$$\begin{aligned} & q_0 = \tfrac{1}{2}\sqrt{t+1}, \qquad t = x_x + y_y + z_z; \\ & q_1 = \tfrac{(y_z - z_y)}{4\, q_0}; \qquad q_2 = \tfrac{(z_x - x_z)}{4\, q_0}; \qquad q_3 = \tfrac{(z_y - y_x)}{4\, q_0} \end{aligned} \tag{B.6}$$

which are valid for $q_0 \neq 0$; while for $q_0 = 0$ the following relations hold

$$\begin{aligned} & \begin{cases} 4 q_2 q_3 = y_z + z_y \\ 4 q_1 q_3 = z_x + x_z \\ 4 q_1 q_2 = y_x + x_y \end{cases} \\ & q_1 = \pm\sqrt{\frac{x_x + 1}{2}} \qquad q_2 = \pm\sqrt{\frac{y_y + 1}{2}} \qquad q_3 = \pm\sqrt{\frac{z_z + 1}{2}} \end{aligned} \tag{B.7}$$

A sequence of rotations can be represented by the product of two quaternions: similarly to $R_t = R(\boldsymbol{q}_2) R(\boldsymbol{q}_1)$, the quaternion $\boldsymbol{q}_t$, expressing the final rotation obtained by the sequence of a rotation $\boldsymbol{q}_1$ followed by a rotation $\boldsymbol{q}_2$, is obtained as

$$\boldsymbol{q}_t = \boldsymbol{q}_2\, \boldsymbol{q}_1 \tag{B.8}$$

We denote the quaternion conjugate of $\boldsymbol{q}$ as $\boldsymbol{q}^*$, with $\boldsymbol{q}^* = q_0 - q_1 i - q_2 j - q_3 k$. This corresponds to an inverse rotation and in fact one has:

$$R(\boldsymbol{q}^*) = R(\boldsymbol{q})^T = R(\boldsymbol{q})^{-1} \tag{B.9}$$

Quaternions, as well as rotation matrices define an algebraic structure over a differential manifold with properties of closure, associativity, presence

of identity element and inverse element, hence they belong to a *Lie group*. The Lie group of rotation matrices is called $\mathsf{SO}(3)$, the special orthogonal group of orthogonal matrices with det = 1. The Lie group of unimodular quaternions $\mathbb{H}_1$ is topologically isomorphic to the S^3 sphere, as such it is a double cover of $\mathsf{SO}(3)$, hence two distinct opposite quaternions represent the same rotation matrix $R \in \mathsf{SO}(3)$.

Given a $\gamma(t) : \mathbb{R} \rightarrow \mathsf{SO}(3)$ be a one parameter sub-group of $\mathsf{SO}(3)$ for which $\gamma(0) = I$, an *exponential map* is defined as $\exp(\Theta) = \gamma(1)$, where $\Theta \in \mathfrak{so}(3)$ is an element of the underlying Lie algebra.

Rotation matrices can be built as exponential maps from elements $\Theta \in \mathfrak{so}(3)$ and vice versa:

$$R = \exp(\Theta) = R(\boldsymbol{u}, \varphi) \tag{B.10}$$

$$\Theta = \log(R) = \underline{\boldsymbol{u}}\varphi \tag{B.11}$$

The *rotation vector* $\boldsymbol{\theta} \in \mathbb{R}^3$, defined for a finite rotation of angle φ about a unit vector $\boldsymbol{u}$, is related to the 3x3 skew-symmetric matrix $\Theta \in \mathfrak{so}(3)$ via

$$\boldsymbol{\theta} = \text{axis}(\Theta) = \boldsymbol{u}\varphi \tag{B.12}$$

$$\Theta = \text{skew}(\boldsymbol{\theta}) = \underline{\boldsymbol{\theta}} = \underline{\boldsymbol{u}}\varphi \tag{B.13}$$

Similarly, the exponential map of unit quaternions links $\mathbb{H}_1$ to its Lie algebra $\mathsf{Im}(\mathbb{H})$ of *pure quaternions* $\boldsymbol{\rho} = [0, \boldsymbol{\nu}]$:

$$\boldsymbol{q} = \exp(\boldsymbol{\rho}) \tag{B.14}$$

$$\boldsymbol{\rho} = \log(\boldsymbol{q}) \tag{B.15}$$

The exponential map (B.14) can be explicitly computed using the closed-form expression $\exp([s, \boldsymbol{v}]) = e^s \left[\cos(||\boldsymbol{v}||), \frac{\boldsymbol{v}}{||\boldsymbol{v}||}\sin(||\boldsymbol{v}||)\right]$, hence:

$$\exp([0, \boldsymbol{\nu}]) = \left[\cos(||\boldsymbol{\nu}||), \frac{\boldsymbol{\nu}}{||\boldsymbol{\nu}||}\sin(||\boldsymbol{\nu}||)\right] \tag{B.16}$$

$$\exp([0, \boldsymbol{u}\xi]) = [\cos(\xi), \boldsymbol{u}\sin(\xi)] \quad \text{for } ||\boldsymbol{u}|| = 1 \tag{B.17}$$

We can use the pure() and imag() operators to convert pure quaternions $[0, \boldsymbol{\nu}]$ from and to rotation vectors $\boldsymbol{\theta}$ by observing that $\xi = \frac{1}{2}\varphi$, $\boldsymbol{\nu} = \frac{1}{2}\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = 2\,\text{imag}(\boldsymbol{\rho}) \tag{B.18}$$

$$\boldsymbol{\rho} = \tfrac{1}{2}\text{pure}(\boldsymbol{\theta}) \tag{B.19}$$

To pass directly from rotation vector $\boldsymbol{\theta}$ to quaternion $\boldsymbol{q} = [s, \boldsymbol{v}]$, and vice versa, one can use the compact expressions:

$$\boldsymbol{q} = \exp\left(\tfrac{1}{2}\text{pure}(\boldsymbol{\theta})\right) = \left[\cos\left(\frac{||\boldsymbol{\theta}||}{2}\right), \frac{\boldsymbol{\theta}}{||\boldsymbol{\theta}||}\sin\left(\frac{||\boldsymbol{\theta}||}{2}\right)\right] \tag{B.20}$$

$$\boldsymbol{\theta} = 2\,\text{imag}(\log(\boldsymbol{q})) = 2\frac{\boldsymbol{v}}{||\boldsymbol{v}||}\tan^{-1}\left(\frac{||\boldsymbol{v}||}{s}\right) \tag{B.21}$$

Note that the last expression is singular for zero rotations, so when $||\boldsymbol{v}|| < \epsilon$ one can compute it as the simplified expression $\boldsymbol{\theta} = 2\boldsymbol{v}$. Also note that rather than using $\tan^{-1}()$ it is advisable to use atan2().

As a special case, the scalar exponential of a quaternion boils down to:

$$\begin{aligned} \boldsymbol{q}^{\,t} &= [\cos(\xi), \boldsymbol{u}\sin(\xi)]^t \\ \boldsymbol{q}^{\,t} &= [\cos(\xi t), \boldsymbol{u}\sin(\xi t)] \end{aligned} \tag{B.22}$$

Finally, one can see that the interpolation method of Eq.6 can be expressed also via a double product of quaternions:

$$\boldsymbol{q} = \boldsymbol{q}_{01}\exp\left(\tfrac{1}{2}\text{pure}(\boldsymbol{u}_1\alpha)\right)\exp\left(\tfrac{1}{2}\text{pure}(\boldsymbol{u}_2\beta)\right) \tag{B.23}$$

or equivalently as $\boldsymbol{q} = \boldsymbol{q}_{01}\exp\left(\alpha\log(\boldsymbol{q}_{01}^*\boldsymbol{q}_{02})\right)\exp\left(\beta\log(\boldsymbol{q}_{02}^*\boldsymbol{q}_{03})\right)$.

Appendix B.2. Velocity and acceleration

Introducing the vector of angular velocity $\boldsymbol{\omega}_e$ expressed in absolute (extrinsic) coordinates, one has

$$\dot{R} = \underline{\boldsymbol{\omega}}_e R \tag{B.24}$$

Similarly, by a second differentiation, and introducing the (extrinsic) angular acceleration $\dot{\boldsymbol{\omega}}_e$, one has

$$\ddot{R} = \underline{\dot{\boldsymbol{\omega}}}_e R + \underline{\boldsymbol{\omega}}_e^2 R \tag{B.25}$$

Given quaternion derivatives, it is possible to compute extrinsic angular velocity and extrinsic angular acceleration, or vice versa, using the following relations:

$$\boldsymbol{\omega}_e = 2\ F(q^*)\dot{\boldsymbol{q}} \tag{B.26}$$

$$\dot{\boldsymbol{\omega}}_e = 2\ F(q^*)\ddot{\boldsymbol{q}} \tag{B.27}$$

$$\dot{\boldsymbol{q}} = \tfrac{1}{2}F(q^*)^T\boldsymbol{\omega}_e \tag{B.28}$$

$$\ddot{\boldsymbol{q}} = \tfrac{1}{2}F(\dot{q}^*)^T\boldsymbol{\omega}_e + \tfrac{1}{2}F(q^*)^T\dot{\boldsymbol{\omega}}_e \tag{B.29}$$

with the introduction of the matrix

$$F(q) = \begin{bmatrix} +q_1 & +q_0 & +q_3 & -q_2 \\ +q_2 & -q_3 & +q_0 & +q_1 \\ +q_3 & +q_2 & -q_1 & +q_0 \end{bmatrix} \tag{B.30}$$

Appendix C. Other interpolation schemes

In the following, for completeness, we briefly describe other interpolation methods that are popular in literature and which we used in our benchmarks for comparisons.

Appendix C.1. SLERP

The spherical linear interpolation (SLERP [24]), with domain $t \in [0, 1]$, interpolates two quaternions $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$ using the exponential formula (B.22) to compute:

$$\begin{aligned} \boldsymbol{q}(t) &= f_{\text{SLERP}}(\boldsymbol{q}_a, \boldsymbol{q}_b, t) \\ &= (\boldsymbol{q}_b \boldsymbol{q}_a^*)^t \, \boldsymbol{q}_a \end{aligned} \tag{C.1}$$

The norm of the quaternion is guaranteed to be unit length. The rotation matrix $R(t)$, if needed, can be computed from $\boldsymbol{q}(t)$ using (B.5).

Appendix C.2. SQUAD

This method, also presented by Shoemake [25], performs a cubic interpolation with domain $t \in [0, 1]$ passing exactly at two extreme attitudes expressed by rotation quaternions $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$, and approaching two intermediate control quaternions $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ similarly to cubic Bezier curves. It does this by using nested calls to the SLERP function, that is:

$$\begin{aligned} \boldsymbol{q}(t) &= f_{\text{SQUAD}}(\boldsymbol{q}_1, \boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{q}_2, t) \\ &= f_{\text{SLERP}}(f_{\text{SLERP}}(\boldsymbol{q}_1, \boldsymbol{q}_2, t), f_{\text{SLERP}}(\boldsymbol{p}_1, \boldsymbol{p}_2, t), 2t(1-t)) \end{aligned} \tag{C.2}$$

In our implementation, an arbitrary sequence of $\boldsymbol{q}_i$ quaternions can be provided, to be interpolated exactly: for each span, the two intermediate $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ quaternions are computed to ensure C^2 continuity across span using the following formula: assuming the i-th span with quaternions $\{\boldsymbol{q}_i, \boldsymbol{p}_i, \boldsymbol{p}_{i+1}, \boldsymbol{q}_{i+1}\}$, we compute

$$\boldsymbol{p}_i = \boldsymbol{q}_i \exp\left(-\frac{1}{4}\left(\log(\boldsymbol{q}_i^* \boldsymbol{q}_{i+1}) + \log(\boldsymbol{q}_i^* \boldsymbol{q}_{i-1})\right)\right) \tag{C.3}$$

Appendix C.3. Quaternion B-splines

For completeness in the following we report the formulation of quaternion B-splines, that was put forward by Kim et al. in the nineties [9], and that is used in our benchmarks for comparison with our method. We assumes a sequence of n given quaternion control points $\boldsymbol{q}_i$, and a knot vector with t_i knots. We assume that conventional B-spline basis functions $B_{i,k}(t)$ can be computed as $(k-1)$-order piecewise polynomials with non-zero support in $[t_i, t_{i+k}]$. Defining the cumulative form of the basis:

$$\tilde{B}_{i,k}(t) = \sum_{j=1}^{n} B_{j,k}(t) \tag{C.4}$$

one can compute the interpolated quaternion with the expression

$$\boldsymbol{q}(t) = \boldsymbol{q}_0^{\tilde{B}_{0,k}(t)} \prod_{i=1}^{n} \exp\left(\log(\boldsymbol{q}_{i-1}^{*}\boldsymbol{q}_i)\tilde{B}_{i,k}(t)\right) \tag{C.5}$$

This method produces an interpolation that is C^{k-2} continuous. For $k = 2$ it boils down to the simple SLERP interpolation, but for higher orders, differently from the other interpolation schemes presented in this paper, it does not pass through the $\boldsymbol{q}_i$ values (except if knot multiplicity is enforced, for instance at the beginning and at the end of the curve).

Appendix C.4. Triplets of angle functions

Assuming that one computes a rotation matrix as a sequence of three rotations about three axis, one has

$$R = R(\alpha, \beta, \gamma) = R(\gamma)R(\beta)R(\alpha) \tag{C.6}$$

where $R(\alpha)$, $R(\beta)$ and $R(\gamma)$ are matrices of rotation about the X, Y, or Z orthogonal axes, in whatever user predefined order. If one provides three scalar functions for the three angles $\alpha(t)$, $\beta(t)$, $\gamma(t)$, one can obtain

$$R(t) = R(\gamma(t))R(\beta(t))R(\alpha(t)) \tag{C.7}$$

Similarly, if using quaternions, one would compute the double product $\boldsymbol{q}(t) = \boldsymbol{q}(\gamma(t))\boldsymbol{q}(\beta(t))\boldsymbol{q}(\alpha(t))$. For instance, $\alpha(t)$ could be a B-spline or Bezier interpolation between α_i corresponding to n keyframe rotations, and similarly for $\beta(t)$, $\gamma(t)$. In any case, whatever the choice of ordering of the sequence of the three axis, there is always a singular attitude for whom, passing close to it, curves may degenerate in sharp and unpredictable oscillations. For this reason, interpolations based on triplets of angle functions are seldom used.

Appendix D. Polynomial motion law

In this section we summarize the quantities needed to define a five-degree polynomial motion law with assigned initial and final conditions of position, velocity and acceleration. If we name

s_i, s_f : initial and final position

v_i, v_f : initial and final velocity

a_i, a_f : initial and final acceleration

$\Delta s = s_f - s_i$

and we impose the following relations

$$\begin{aligned} A &= s_i \\ B &= v_i \\ C &= \frac{a_i}{2} \\ D &= \frac{a_f - 3a_i}{2T} - \frac{6v_i + 4v_f}{T^2} + \frac{10\Delta s}{T^3} \\ E &= \frac{3a_i - 2a_f}{2T^2} + \frac{8v_i + 7v_f}{T^3} - \frac{15\Delta s}{T^4} \\ F &= \frac{a_f - a_i}{2T^3} - \frac{3(v_i + v_f)}{T^4} + \frac{6\Delta s}{T^5} \end{aligned} \tag{D.1}$$

we may formulate the desired motion law and its derivatives as follows:

$$\begin{aligned} s(t) &= Ft^5 + Et^4 + Dt^3 + Ct^2 + Bt + A \\ \dot{s}(t) &= 5Ft^4 + 4Et^3 + 3Dt^2 + 2Ct + B \\ \ddot{s}(t) &= 20Ft^3 + 12Et^2 + 6Dt + 2C \end{aligned} \tag{D.2}$$

References

[1] A. Barr, B. Currin, S. Gabriel, and J. Hughes. Smooth interpolation of orientations with angular velocity constraints using quaternions. *ACM Siggraph Computer Graphics*, 26(2):313–320, 1992.

[2] M. Basarab. Interpolating solid orientations with quaternion curves based on atomic functions and cardinal series. *Science & Military Journal*, 5(2):87, 2010.

[3] O. Bauchau and S. Han. Interpolation of rotation and motion. *Multibody System Dynamics*, 31(3):339–370, 2014.

[4] V. Bolotnikov. Polynomial interpolation over quaternions. *Journal of Mathematical Analysis and Applications*, 421(1):567–590, 2015.

[5] E. Dam, M. Koch, and M. Lillholm. *Quaternions, interpolation and animation*, volume 2. Citeseer, 1998.

[6] S. Gabriel and J. Kajiya. Spline interpolation in curved space. State of the art in image synthesis. Siggraph course notes, 1985.

[7] S. Ghosh and D. Roy. Consistent quaternion interpolation for objective finite element approximation of geometrically exact beam. *Computer Methods in Applied Mechanics and Engineering*, 198(3-4):555–571, 2008.

[8] W.R. Hamilton. On quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 25(163):10–13, 1844.

[9] M.-J. Kim, M.-S. Kim, and S.Y. Shin. A C^2-continuous b-spline quaternion curve interpolating a given sequence of solid orientations. In *Proceedings Computer Animation*, pages 72–81. IEEE, 1995.

[10] M.-J. Kim, M.-S Kim, and S.Y. Shin. A general construction scheme for unit quaternion curves with simple high order derivatives. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 369–376, 1995.

[11] M.-J. Kim, M.-S. Kim, and S.Y. Shin. A compact differential formula for the first derivative of a unit quaternion curve. *The Journal of Visualization and Computer Animation*, 7(1):43–57, 1996.

[12] G. Legnani, F. Casolo, P. Righettini, and B. Zappa. A homogeneous matrix approach to 3d kinematics and dynamics - I, theory. *Mechanism and Machine Theory*, 31(5):573–587, 1996.

[13] G. Legnani, F. Casolo, R. Righettini, and B. Zappa. A homogeneous matrix approach to 3d kinematics and dynamics - II, applications to chains of rigid bodies and serial manipulators. *Mechanism and Machine Theory*, 31(5):589–605, 1996.

[14] G. Legnani and I. Fassi. Representation of 3d motion by projective angles. In *Proceedings of the ASME Design Engineering Technical Conference*, volume 5C-2015, 2015.

[15] G. Legnani and I. Fassi. Kinematics analysis of a class of spherical pkms by projective angles. *Robotics*, 7(4), 2018.

[16] P. Li, R. Guo, P. Wang, and Y. Huang. Research on tool path planning for five-axis machining. In *2009 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 2338–2342. IEEE, 2009.

[17] M. Neubauer and A. Müller. Smooth orientation path planning with quaternions using b-splines. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2087–2092. IEEE, 2015.

[18] D. Pletinckx. Quaternion calculus as a basic tool in computer graphics. *The Visual Computer*, 5(1-2):2–13, 1989.

[19] Y. Pu, Y. Shi, X. Lin, Y. Hu, and Z. Li. C2-continuous orientation planning for robot end-effector with b-spline curve based on logarithmic quaternion. *Mathematical Problems in Engineering*, 2020, 2020.

[20] R. Ramamoorthi and A. Barr. Fast construction of accurate quaternion splines. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 287–292, 1997.

[21] B. Ravani and F.C. Park. Bézier curves on riemannian manifolds and lie groups with kinematic applications. *Journal of Mechanical Design*, 117(1):36–40, 1995.

[22] I. Romero. The interpolation of rotations and its application to finite element models of geometrically exact rods. *Computational mechanics*, 34(2):121–133, 2004.

[23] J. Schlag. Using geometric constructions to interpolate orientation with quaternions. In *Graphics Gems II*, pages 377–380. Elsevier, 1991.

[24] K. Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, 1985.

[25] K. Shoemake. Quaternion calculus and fast animation. In *SIGGRAPH 1987*, 1987.

[26] J. Tan, Y. Xing, W. Fan, and P. Hong. Smooth orientation interpolation using parametric quintic-polynomial-based quaternion spline curve. *Journal of Computational and Applied Mathematics*, 329:256–267, 2018.

[27] A. Tasora. Quaternioni. In E. Cheli and E. Pennestri, editors, *Cinematica e Dinamica dei Sistemi Multibody*, pages 117–137. CEA Casa Editrice Ambrosiana, 2006.

[28] A. Tasora, R. Serban, H. Mazhar, A. Pazouki, D. Melanz, J. Fleischmann, M. Taylor, H. Sugiyama, and D. Negrut. Chrono: An open source multi-physics dynamics engine. In *International Conference on High Performance Computing in Science and Engineering*, pages 19–49. Springer, 2015.

[29] Z. Yongguo, H. Xiang, F. Wei, and L. Shuanggao. Trajectory planning algorithm based on quaternion for 6-dof aircraft wing automatic position and pose adjustment method. *Chinese Journal of Aeronautics*, 23(6):707–714, 2010.