Survey of Higher Order Rigid Body Motion Interpolation Methods for Keyframe Animation and Continuous-Time Trajectory Estimation

Adrian Haarbach [1,2] Tolga Birdal [1,2] Slobodan Ilic [1,2]
[1] Technische Universitat München, Germany [2] Siemens AG, Munich, Germany
adrian.haarbach@tum.de tolga.birdal@tum.de slobodan.ilic@siemens.com

Abstract

In this survey we carefully analyze the characteristics of higher order rigid body motion interpolation methods to obtain a continuous trajectory from a discrete set of poses. We first discuss the tradeoff between continuity, local control and approximation of classical Euclidean interpolation schemes such as Bézier and B-splines. The benefits of the manifold of unit quaternions $\mathbb{SU}(2)$, a double-cover of rotation matrices $\mathbb{SO}(3)$, as rotation parameterization are presented, which allow for an elegant formulation of higher order orientation interpolation with easy analytic derivatives, made possible through the Lie Algebra $\mathfrak{su}(2)$ of pure quaternions and the cumulative form of cubic B-splines. The same construction scheme is then applied for joint interpolation in the full rigid body pose space, which had previously been done for the matrix representation $\mathbb{SE}(3)$ and its twists, but not for the more efficient unit dual quaternion $\mathbb{DH}_1$ and its screw motions. Both suffer from the effects of coupling translation and rotation that have mostly been ignored by previous work. We thus conclude that split interpolation in $\mathbb{R}^3 \times \mathbb{SU}(2)$ is preferable for most applications. Our final runtime experiments show that joint interpolation in $\mathbb{SE}(3)$ is 2 times and in $\mathbb{DH}_1$ 1.3 times slower - which furthermore justifies our suggestion from a practical point of view.

1. Introduction

Choosing a suitable trajectory representation to model the 3D motion of a rigid body is an important design decision of interpolation, filtering and optimization techniques in graphics, vision and robotics. Compared to a discrete set of dense poses, higher order methods produce a smooth, time-continuous curve by weighted summation of a sparse set of base poses acting as temporal basis functions. Combined with a suitable orientation parameterization, the resulting curve must fulfill a few important properties to be useful for keyframe animation in graphics and continuous-time trajectory estimation and optimization in robotics:

- *Local control* so a pose update has bounded influence.
- $\mathcal{C}^2$ *continuity* fulfilling physical smothness constraints.
- *No singularities* to globally represent all orientations.
- *Few parameters* to allow for efficient computation.
- *Analytic derivatives* to be able to synthesize angular velocity and linear acceleration measurements.

However, choosing a representation is not easy because there is no perfect solution: 1) The interpolation method is always [26] a trade-off between interpolation vs. approximation, high continuity, local vs. global control and computational complexity. 2) There is no minimal rotation parameterization without singularities [10]. Every additional parameter used in a singularity-free representation incurs additional constraints on the parameters that need to be maintained during interpolation and optimization.

In this paper we compare many existing methods for camera trajectory interpolation providing an intuitive visual and experimental comparison among them and highlighting insights that are not obvious. In the end we furthermore briefly introduce two novel formulations (ScFus, DLFus) of higher order dual quaternion interpolation as efficient alternatives to the usual joint interpolation with $\mathbb{SE}(3)$ matrices.

Since animation techniques are best explored interactively, we provide our web and C++ applications to interact with and visualize Euclidean, orientation and rigid body motion interpolation methods that produced all our figures and runtime evaluations as well as an overview table of all methods: http://adrian-haarbach.de/interpolation-methods.

The paper is organized as follows: After the related work (Sec. 2), we will first discuss the confronting requirements of local control, continuity and interpolation vs. approximation of the two most popular higher order interpolation methods (Sec. 3) in Euclidean space. A comparison of different orientation interpolation methods (Sec. 4) follows. Then we compare approaches for the full rigid body motion interpolation (Sec. 5). After a quick experimental runtime evaluation of the different methods (Sec. 6), we briefly conclude our findings (Sec. 7).

2. Related Literature

Orientation interpolation methods were first explored in graphics due to the need to interpolate smoothly between camera frames for animation. The rotation parameters were usually interpolated in Euclidean space, with subsequent re-orthogonalization in case of rotation matrices or renormalization in case of unit quaternions, or alternatively, Euler angles were used [4]. Out of these three options, only **Q**uaternion **l**inear **b**lending (QLB) (9) (QLERP in [18]) follows the shortest path between the keyframes in rotation space, but at the cost of non-constant rotational speed due to the uneven angular spacing of the intermediate quaternions resulting from the renormalization [12].

The correct way to interpolate between two unit quaternions with constant rotational speed is on the surface of the sphere $\mathbb{S}^3$, discovered by Shoemake [28] in 1985 and thus termed **S**pherical **l**inear int**erp**olation (SLERP) (14). Two years later, the same author [29] extends his idea to higher order interpolation of sequences of unit quaternions. Using *bilinear parabolic blending* of 4 base quaternions located at the corners of a quadrangle and a special computation for the positions of the inner control points, it is possible to gain $\mathcal{C}^1$ continuity. Thus this geometric construction method was termed **S**pherical cubic spline **quad**rangle (SQUAD) (15).

Another 8 years later, in 1995, Kim *et al*. [21] took an important step towards even higher order orientation interpolation by extending the algebraic construction methods of spline (*e.g*. Hermite, Bézier, B-spline) curves from $\mathbb{R}^3$ to $\mathbb{SO}(3)$. The key to their success lies in the usage of the manifold structure of unit quaternions $\mathbb{H}_1$ and the transformation of Euclidean interpolation methods to cumulative basis form, allowing to formulate a curve in $\mathbb{SO}(3)$ as a product of simple unit quaternion curves, each representing the relative orientation difference between neighbouring key frames. Of particular interest is their cubic **Cu**mulative **B-sp**line curve (CuBsp) (17) since it allows for $\mathcal{C}^2$ continuity with local control.

Rigid body motion interpolation methods based on this idea however took another 18 years to be transfered from graphics to vision and robotics by Lovegrove *et al*. through the **Sp**line **Fus**ion twist curve (SpFus) [22, 27], which the authors use for continuous-time estimation in rolling shutter camera calibration for SfM and visual-inertial SLAM. As a continuous-time trajectory representation, they use CuBsp, but applied to $\mathbb{SE}(3)$ instead of quaternions. They justify the choice of this joint parameterization because it models torque-minimal trajectories. However, its inherent coupling of translation and rotation is suboptimal for this kind of application, a fact first mentioned briefly in 2014 in Forssen's lecture slides [7], one year after Spline Fusion [22] first appeared. This argument is only very recently, in 2018, expanded upon in the upcoming journal article by Ovren [25].

Split parameterization of rotation and translation, as usually done in graphics, furthermore avoids the costly matrix multiplications necessary for $\mathbb{SE}(3)$ trajectory evaluations. Efficient quaternion algebra [21] can be used instead on the rotation curve. When implementing analytic derivatives efficiently *e.g*. (19), this splitting becomes even more important to avoid superfluous computations of uneccessary second-order derivatives of the rotation part.

Continuous-time estimation theory predates Spline Fusion [22], since it was introduced to the robotics domain in 2012, in particular to move from discrete to continuous-time simultaneous localization and mapping (CT-SLAM) [8], with the following benefits: 1) High-rate sensors such as an inertial measurement unit (IMU) capture a lot of data, which in discrete estimation methods would require to include a pose variable in the state for each measurement, making it very large. 2) Continuously capturing devices such as **li**ght **d**etection **a**nd **r**anging (LiDAR) scanners or rolling shutter cameras, when moved during acquisition, produce distortion artefacts if their measurements are handled as discrete snapshots in time. However, even in their subsequent journal article [9] which appeared after Spline Fusion [22], the authors interpolate rotation in Euclidean space using a non-cumulative B-spline of Cayley-Gibbs-Rodrigues vector coefficients, which has singularities and which results in non-constant rotational speed between two keyframes, making the synthesised IMU measurements unstable. A more promising direction is taken by the journal article [30], extending the theory of the cumulative cubic B-spline unit quaternion curves [21] to general Lie groups and arbitrary spline order. The most surprising discovery in the experiments was the strong influence of the spline order on the expressiveness of the curve.

Dual quaternion approaches did not receive the same attention in continuous-time estimation, even though they were used successfully for rigid transformation blending in skinning [15, 16, 17], where SLERP was extended to **Sc**rew **l**inear int**erp**olation (ScLERP), and for a fast approximation, QLB to **D**ual quaternion **l**inear **b**lending (DLB). Only recently, their usefulness for inter- and extrapolation [3] and camera pose filtering [2] is shown, with an extensive treatment of their differential geometry nature. However, to the best of our knowledge, they were so far only used to define piecewise $\mathcal{C}^0$ curves, not inside higher-order construction schemes. Geometrically, dual quaternions and their tangent representation as screw motions [5] are equivalent to the $\mathbb{SE}(3)$ matrices and $\mathfrak{se}(3)$ twists, meaning they exhibit the same coupling of rotation and translation, but the storage size is lower and, even more importantly, their exponential and logarithmic maps can be more efficiently implemented.

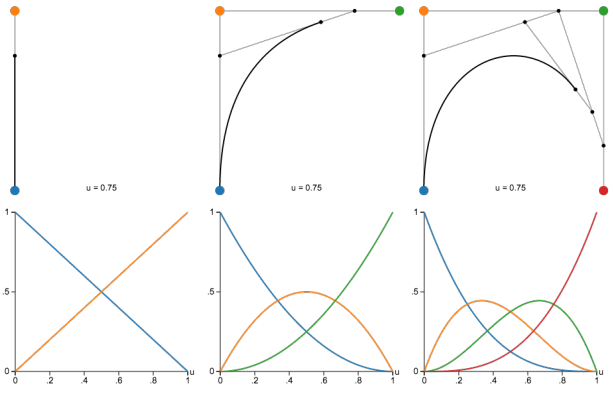
3. Euclidean interpolation methods

Linear interpolation (LERP) The straight line connecting $\mathbf{p}_0, \mathbf{p}_1 \in \mathbb{R}^x$ with $u \in [0, 1]$ is given by:

$$\text{LERP}(\mathbf{p}_0, \mathbf{p}_1, u) = (1-u)\mathbf{p}_0 + u\mathbf{p}_1 \tag{1}$$

For more points, piecewise linear interpolation results in straight line segments for each consecutive pair of points. Piecewise curves interpolate all points, have local control and low complexity, but are only $\mathcal{C}^0$ continuous, meaning that the velocity is piecewise continuous and acceleration is infinite at the control points and 0 elsewhere. Thus, they cannot represent smooth motions.

Bézier curve To gain higher order continuity, one can apply (1) to pairs of $n+1$ consecutive control points iteratively. This geometric construction scheme (Fig. 1) by *repeated linear interpolation* is known as the **de Casteljau** algorithm (Mortenson [24] ch.4), recursively defined by $\mathbf{p}_i^{(0)} = \mathbf{p}_i$ and $\mathbf{p}_i^{(k)} = (1-u)\mathbf{p}_i^{(k-1)} + u\mathbf{p}_{i+1}^{(k-1)}$ for $i \in [0, n-k], k \in [1, n]$, resulting in a curve of degree n. The weights of the control points $\mathbf{p}_i$ are exactly the **Bernstein polynomials** $B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}$, $\mathbf{p}(u) = \sum_{i=0}^{n} B_i^n(u)\mathbf{p}_i$ (Farin [6] ch.5) which allow for an explicit curve representation in matrix form (Parent [26] p.460). A Bézier curve interpolates the first and last control point, while the inner ones are approximated. As a polynomial, a Bézier curve is easily differentiable. However, the degree n of the curve grows linearly with the number of control points $n+1$. High degree curves suffer from the oscillation problems inherent to high order polynomials and also have a high complexity. Additionally, they have global control, meaning a base pose update affects the whole curve.



(a) linear (b) quadratic (c) cubic

Figure 1: De Casteljau construction (top) of Bézier curves of degree 1, 2 and 3 evaluated at $u = 0.75 \in [0, 1]$. Control points are associated by colour with their respective weight, given by the Bernstein polynomials basis of the corresponding degree (bottom).

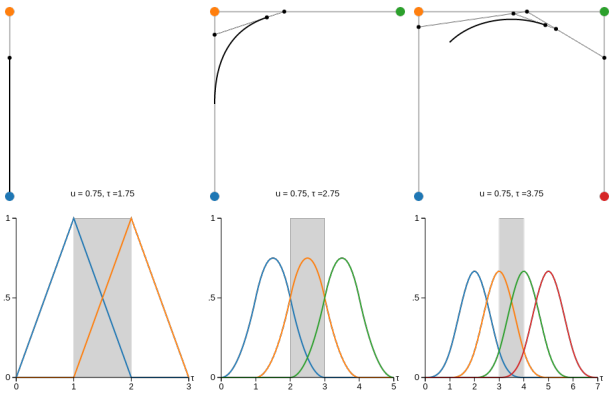
B-spline is a piecewise polynomial function and a generalization of a Bézier curve. It is defined over a *knot sequence* $\{\tau_i\}, \tau_i \leq \tau_{i+1}$, each knot associated to a control point $\mathbf{p}_i$, which allows to decouple the degree $k-1$ of the curve from the number $n+1$ of control points, in contrast to the Bézier curve. The B-spline curve $\mathbf{p}(\tau) = \sum_{i=0}^{n} N_i^k(\tau)\mathbf{p}_i$ is a weighted sum of **B-spline basis** functions N_i^k (Farin [6] ch.8), which can be computed (Fig. 2) using the **de Boor** algorithm (Mortenson [24] ch.5), recursively defined by $N_i^0(\tau) = \mathbf{1}_{\tau \in [\tau_i, \tau_{i+1})}$ and $N_i^k(\tau) = (1-\alpha_{i,k})N_{i-1}^{k-1}(\tau) + \alpha_{i,k}N_i^{k-1}(\tau)$ with $\alpha_{i,k} = \frac{\tau - \tau_i}{\tau_{i+n+1-k} - \tau_i}$, where the weights $\alpha_{i,k}$ change at each iteration. A new global time variable $\tau \in [\tau_0, ..., \tau_{n+1}]$ is introduced while the parametric variable $u = \frac{\tau - \tau_i}{\tau_{i+1} - \tau_i} \in [0, 1]$ with $\tau \in [\tau_i, \tau_{i+1})$ interpolates within a B-spline segment between two knots. $\mathcal{C}^2$ continuous curves require to use at least cubic B-splines while a uniform knot spacing allows to give an explicit matrix form as follows (Parent [26] p. 464):

$$\mathbf{p}(\tau) = [u^3, u^2, u, 1]\frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \tag{2}$$

A B-spline has local support of k control points. The complexity is thus bounded by the continuity requirement, and not by the number of control points. It does not interpolate, but only approximates the control points. In a cubic B-spline, four consecutive control points influence the curve at time τ, where $\tau \in [\tau_i, \tau_{i+1})$. In contrast to a Bézier curve, this means that the curve is only defined in $[\tau_1, \tau_n]$ instead of $[\tau_0, \tau_{n+1}]$. To cope with this issue *phantom points* have to be added at the beginning and end of the spline so that the curve is defined over the timespan of all control points. These can be placed so that the B-spline actually interpolates the first and last control point (Fig. 4d).

Continuity vs. local control vs. approximation When interpolating many control points with $\mathcal{C}^2$ continuity requirements, we can either use a B-spline or a *composite* Bézier curve, which is a series of Bézier segments, where the last control point of one segment is the first control point of the next. This ensures $\mathcal{C}^0$ continuity. For higher order continuity, we have to place interior control points in a special way, because in contrast to B-spline basis functions (Fig. 3e), the Bézier basis functions (Fig. 3a) of one segment have no influence into neighboring ones.

$\mathcal{C}^2$ continuity requires at least cubic curves. However, to gain $\mathcal{C}^2$ continuity, a cubic composite Bézier curve loses local control. To enforce it, all the control points become dependent on each other (Fig. 3c). The constraints on interior control point placement propagate through the whole curve, so if a single control point moves, the whole curve needs



(a) linear (b) quadratic (c) cubic

Figure 2: B-spline segment of degree 1,2 and 3 constructed with the de Boor algorithm. The gray area indicates the current segment $u \in [0, 1]$ between the two knots τ_i, τ_{i+1}. Only the basis functions which influence the current segment are plotted, notice how they extend to neighbouring segments to ensure continuity across segment boundaries. The width or support of a basis function is exactly degree+1 segments.

to be re-evaluated. On the other hand, cubic B-splines have $\mathcal{C}^2$ continuity and local control, but they lose the interpolation property of a composite cubic Bézier curve, they only approximate the control points (Fig. 3d).

For keyframe animation, an interpolation method should interpolate the given sequence, but for continuous-time trajectory estimation and filtering methods, an approximating curve converging to the desired shape during optimization is sufficient and might be even beneficial due to its intrinsic handling of outliers. Local control allows that only a subset of control points has to be updated when new measurements arrive and the curve automatically stays $\mathcal{C}^2$ continuous.

Summary

- A $\mathcal{C}^2$ continuous composite cubic Bézier curve **interpolates** the control points but has **global control**.
- A $\mathcal{C}^2$ continuous cubic B-spline has **local control** but only **approximates** the control points.

4. Orientation interpolation methods

In 1998, Dam *et al.* [4] establish a framework to compare orientation interpolation methods. A trajectory in $\mathbb{SO}(3)$ on the provided orientations (Tab. 1) lives on one hemisphere of the usual sphere $\mathbb{S}^2$ instead of $\mathbb{S}^3$, because $z = 0$, so it can be printed to paper (Fig. 4). The angular velocity norm plot furthermore helps to judge the curve's smoothness. Regarding the orientation parameterization, one can usually choose between Euler angles, rotation matrices or unit quaternions, but no matter the choice, one either has to deal with singularities or over-parameterization. For interpolation, the last one is however suited best, because it is singularity-free at

i	$\theta \in [-\pi, \pi]$	$\boldsymbol{\omega} \in \mathbb{R}^3$	$\mathbf{q} = [w, x, y, z] \in \mathbb{S}^3$
0	1	(1, 3, 0)	[0.88, 0.15, 0.45, 0]
1	1.9	(-1, 0, 0)	[0.58, -0.81, 0.00, 0]
2	0	(-2, 1, 0)	[1.00, -0.00, 0.00, 0]
3	-2	(3, 4, 0)	[0.54, -0.50, -0.67, 0]
4	-1	(-1, 4, 0)	[0.88, 0.12, -0.47, 0]
5	1	(2, 3, 0)	[0.88, 0.27, 0.40, 0]

Table 1: Key frames with index i, rotation angle θ and axis $\boldsymbol{\omega}$ corresponding to the unit quaternion q adapted from [4, Table 5.1].

the cost of just one additional unit norm constraint.

Quaternions ($\mathbb{H}$) A *quaternion* $\mathbf{q}$ is in essence the extension of a complex number $c = a+bi \in \mathbb{C}, \mathbb{C} = \mathbb{R}+\mathbb{R}i, i^2 = -1$ from 2 to 4 dimensions:

$$\mathbb{H} = \mathbb{C} + \mathbb{C}j, j^2 = -1, ij = -ji \tag{3}$$

Formally, a quaternion $\mathbf{q} \in \mathbb{H}$ may be represented by a vector $\mathbf{q} = [q_w, q_x, q_y, q_z]^T = [q_w, \mathbf{q}_v]^T$ together with the definitions:

$$\text{adjoint/conjugate}: \quad \bar{\mathbf{q}} = [q_w, -\mathbf{q}_v]^T \tag{4}$$

$$\text{norm}: \|\mathbf{q}\| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} \tag{5}$$

$$\text{inverse}: \mathbf{q}^{-1} = \frac{\bar{\mathbf{q}}}{\|\mathbf{q}\|} \tag{6}$$

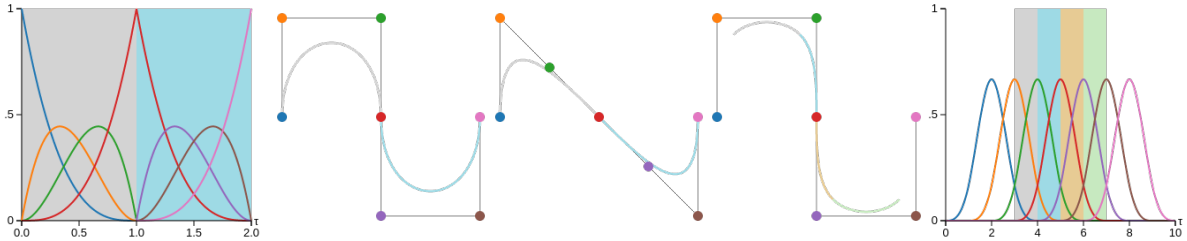
Multiplication of two quaternions is associative and distributive, but non-commutative since $\mathbf{q}\mathbf{q}' \neq \mathbf{q}'\mathbf{q}, ij = -ji$. With dot $\cdot$ and cross $\times$ product it can be defined as [4, 14]:

$$\mathbf{q}\mathbf{q}' = [q_w q'_w - \mathbf{q}_v \cdot \mathbf{q}'_v \ , \ \mathbf{q}_v \times \mathbf{q}'_v + q_w \mathbf{q}'_v + \mathbf{q}_v q'_w] \tag{7}$$

Special unitary group of unit quaternions ($\mathbb{SU}(2)$)

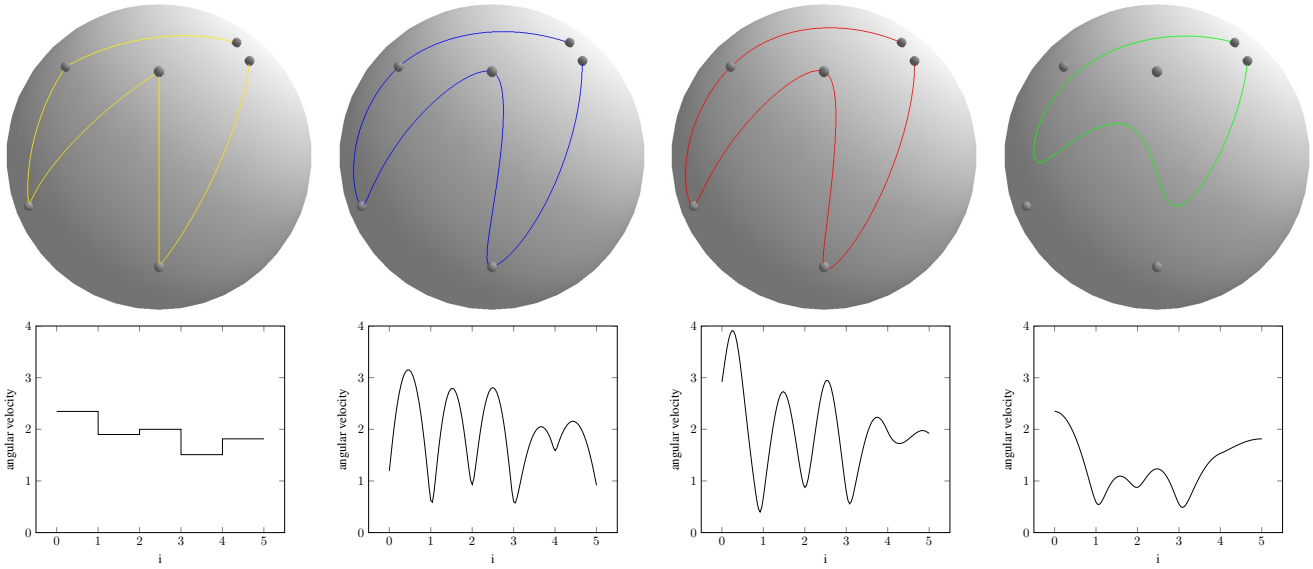
$$\mathbb{SU}(2) \cong \mathbb{S}^3 \cong \mathbb{H}_1 = \{\mathbf{q} \in \mathbb{H} \mid \|\mathbf{q}\| = 1\} \tag{8}$$

Just as complex numbers can be used to represent rotations in $\mathbb{R}^2$ using Euler's formula $\mathrm{e}^{\mathrm{i}\,\varphi} = \cos(\varphi) + \mathrm{i}\,\sin(\varphi)$, certain quaternions can be used to represent rotations in $\mathbb{R}^3$. These are exactly the quaternions with unit norm $\mathbb{H}_1$, a subspace of $\mathbb{H}$. By identifying quaternion coefficients with $\mathbb{R}^4$, unit quaternion coefficients form the 3-dimensional hypersphere $\mathbb{S}^3$. Note that topologically, the groups $\mathbb{SU}(2) \cong \mathbb{S}^3 \cong \mathbb{H}_1$ are all isomorphic to each other. Furthermore, $\mathbb{SU}(2)$ is a universal double cover of $\mathbb{SO}(3)$, meaning that the two antipodal unit quaternions $\mathbf{q}$ and $-\mathbf{q}$ both represent the same orientation. Because of this, the distance measure in $\mathbb{S}^3$ is also twice as long as that of $\mathbb{SO}(3)$. Rotation **composition** can just be expressed by quaternion multiplication (7). Note that for unit quaternions, the **inverse** rotation is simply the quaternion conjugate $\mathbf{q}^{-1} = \bar{\mathbf{q}}$.



(a) composite Bézier curve basis (b) comp. $\mathcal{C}^1$ Bézier curve (c) comp. $\mathcal{C}^2$ Bézier curve (d) $\mathcal{C}^2$ B-spline (e) B-spline basis

Figure 3: Comparison of composite cubic Bézier curves with a uniform cubic B-spline, defined by 7 control points. The segments in the basis function plot are shaded in the same color as the curve segment influenced by these basis functions.



(a) SLERP: The keyframes are interpolated piecewise [28], resulting in local control. However, the curve is only $\mathcal{C}^0$ and thus not differentiable at the keyframes. The angular velocity graph is piecewise continuous, meaning that the angular velocity is constant in between keyframes.

(b) SQUAD: The curve [29] is $\mathcal{C}^1$ and thus at least once differentiable everywhere. The angular velocity graph is continuous and has minima at the keyframes. Changes of one control point only propagate to the immediately neighboring segments, thus we have local control.

(c) RQBez: The curve looks very similar to SQUAD, but is actually a uniform cubic Bézier curve of the quaternion coefficients $\in \mathbb{R}^4$ with subsequent renormalization. The segments are joined with $\mathcal{C}^2$ continuity requirements, thus we have global control.

(d) CuBsp: This $\mathcal{C}^2$ curve is the uniform cumulative cubic B-spline on $\mathbb{SU}(2)$, introduced by [21] and used in [22, 27]. The angular velocity graph shows that this curve minimizes angular accelerations. It has local control, but the inner keyframes are only approximated.

Figure 4: Comparison of orientation interpolation methods on the surface of the sphere $\mathbb{S}^2$ and by the norm of their angular velocity $\mathfrak{su}(2)$.

Quaternion linear blending (QLB) [18] A good approximation to interpolate two orientations is LERP (1) on quaternion coefficients along the 4D straight line chord below the surface of $\mathbb{S}^3$, followed by a renormalization. This results in non-constant rotational velocity along $u \in [0, 1]$:

$$\mathrm{QLB}(\mathbf{q}_0, \mathbf{q}_1, u) = \frac{\mathrm{LERP}(\mathbf{q}_0, \mathbf{q}_1, u)}{\|\mathrm{LERP}(\mathbf{q}_0, \mathbf{q}_1, u)\|} \tag{9}$$

Renormalized quaternion Bézier curve (RQBez) For multiple frames, an Euclidean cubic composite Bézier curve (Fig. 3b) on quaternion coefficients with subsequent renormalization can be used. The segments are joined with (Euclidean) $\mathcal{C}^2$ continuity, which results in an interpolating curve (Fig. 4c) that looks smooth and is fast to evaluate, but which doesn't minimize angular accelerations.

Lie algebra of pure quaternions ($\mathfrak{su}(2)$)

$$\mathfrak{su}(2) \cong \mathbb{R}^3 \cong \mathbb{H}_0 = \{\mathbf{q} \in \mathbb{H} \quad | \quad q_w = 0\} \tag{10}$$

The group of unit quaternions $\mathbb{H}_1$ is isomorphic to the Lie group $\mathbb{SU}(2)$. Thus, its corresponding Lie algebra $\mathfrak{su}(2)$ also has a representation with quaternions. These are exactly the pure quaternions $\mathbb{H}_0$. Since the scalar part of these quaternions is zero, we can identify their vector part with $\mathbb{R}^3$, which can be interpreted as an angular velocity vector [13]. Topologically, the algebras $\mathfrak{su}(2) \cong \mathbb{R}^3 \cong \mathbb{H}_0$ are all isomorphic to each other and form the tangent space at the identity of the Lie group $\mathbb{SU}(2)$.

Quaternion exp and log Similar to matrices [23], one can define exponentiation and logarithm for quaternions. These

allow to convert between Lie group and algebra as follows:

$$\boldsymbol{\omega} \in \mathbb{R}^3 \cong \mathfrak{su}(2) \underset{\log}{\overset{\exp}{\rightleftharpoons}} \mathbf{q} \in \mathbb{S}^3 \cong \mathbb{SU}(2) \tag{11}$$

However, one has to take care of the double covering and the resulting different distance measure when implementing above functions. Given an angular velocity vector $\boldsymbol{\omega} = \theta\hat{\boldsymbol{\omega}} \in \mathbb{R}^3, \theta = \|\boldsymbol{\omega}\|, \hat{\boldsymbol{\omega}} \in \mathbb{S}^2$, the exponential

$$\exp(\boldsymbol{\omega}) = [\cos(\frac{1}{2}\theta), \hat{\boldsymbol{\omega}}\sin(\frac{1}{2}\theta)]^T = [q_w, \mathbf{q}_v]^T = \mathbf{q} \tag{12}$$

becomes the unit quaternion which represents the rotation by angle θ about the axis $\hat{\boldsymbol{\omega}}$ [21]. The exponential map exp can: 1) Be interpreted as a mapping from the angular velocity vector $\boldsymbol{\omega} \in \mathbb{R}^3$ into the unit quaternion which represents the rotation. 2) Be used as a conversion from the angle-axis representation $(\theta, \hat{\boldsymbol{\omega}})$ of rotations to unit quaternions. Omitting the scalar constant 0.5 in above equation yields a rotation by angle 2θ instead of θ around $\hat{\boldsymbol{\omega}}$, because $\boldsymbol{\omega}$ is then measured in $\mathbb{S}^3$ instead of $\mathbb{SO}(3)$. Since cos and sin are periodic functions, the range of arccos is π, and the inverse of exp, called log, can only recover the original angles θ modulo 2π:

$$\log(\mathbf{q}) = 2\arccos(q_w)\frac{\mathbf{q}_v}{\|\mathbf{q}_v\|} = \theta\hat{\boldsymbol{\omega}} = \boldsymbol{\omega} \tag{13}$$

Spherical linear interpolation (SLERP) [28] (Shoemake '85) With these insights, it is possible to define interpolation with constant angular velocity between two quaternions along $u \in [0, 1]$:

$$\text{SLERP}(\mathbf{q}_0, \mathbf{q}_1, u) = \mathbf{q}_0 \exp(u \log(\bar{\mathbf{q}}_0\mathbf{q}_1)) \tag{14}$$

It produces a geodesic in $\mathbb{S}^3$, with the property of constant angular velocity along the corresponding path in $\mathbb{SO}(3)$. This is due to the fact that the intermediate quaternions are equally spaced on this curved surface. If interpolating more than two orientations, a piecewise application is possible (Fig. 4a), but provides only $\mathcal{C}^0$ continuity.

Spherical cubic spline quadrangle (SQUAD) [29] (Shoemake '87) $\mathcal{C}^1$ continuity can be gained using *bilinear parabolic blending* on a quaternion quadrangle, which is nicely explained in [32]. To ensure $\mathcal{C}^1$ continuity across segment boundaries, the two inner control quaternions $\mathbf{s}_i, \mathbf{s}_{i+1}$ are computed in a special way (16) from 3 adjacent quaternions of the keyframe sequence. The composite curve (Fig. 4b) is then $\mathcal{C}^1$ continuous [4, proof of prop. 30] and has local control. Nevertheless, modifying one keyframe requires to recompute interior control points in directly adjacent segments to maintain this continuity across segment boundaries. Since the original paper [29] is not available anymore, we replicate the definition of SQUAD, using quaternion exponentiation (12) and logarithm (13) here according to [4, def. 17]:

$$\begin{aligned}&\text{SQUAD}(\mathbf{q}_i, \mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{q}_{i+1}, u)\\ &= \text{SLERP}\big(\text{SLERP}(\mathbf{q}_i, \mathbf{q}_{i+1}, u),\\ &\qquad \text{SLERP}(\mathbf{s}_i, \mathbf{s}_{i+1}, u), 2u(1-u)\big)\end{aligned} \tag{15}$$

$$\mathbf{s}_i = \mathbf{q}_i \exp\left(-\frac{\log(\mathbf{q}_i^{-1}\mathbf{q}_{i+1}) + \log(\mathbf{q}_i^{-1}\mathbf{q}_{i-1})}{4}\right) \tag{16}$$

Cumulative B-spline curve (CuBsp) [21] (Kim, Kim, Shin '95) A B-spline (2) is given as sums of basis functions with control points as coefficients $\mathbf{p}(\tau) = \sum_{i=0}^{n} N_i^k(\tau)\mathbf{p}_i$. Using the cumulative basis $\tilde{N}_i^k(\tau) = \sum_{j=i}^{n} N_j^k(\tau)$ with the property $\tilde{N}_0^k(\tau) = 1, \tau > \tau_0$ due to the partition of unity, the B-spline can be rearranged into cumulative form as follows [21]: $\mathbf{p}(\tau) = \mathbf{p}_0 + \sum_{i=1}^{n} \tilde{N}_i^k(\tau)(\mathbf{p}_i - \mathbf{p}_{i-1})$. In Euclidean space, the difference between consecutive control points $\mathbf{p}_{i-i}$ and $\mathbf{p}_i$ is the time difference $\Delta\tau = \tau_i - \tau_{i-1}$ between those points times the velocity $\mathbf{v}_i\Delta\tau = -\mathbf{p}_{i-1} + \mathbf{p}_i$. While we cannot apply the B-spline basis form to the Lie group of unit quaternions (because it is not closed under addition) the concept of velocity also exists in quaternion space, more specifically in its associated Lie algebra of pure quaternions, which can be interpreted as angular velocity vectors $\boldsymbol{\omega}_i\Delta\tau = \log(\bar{\mathbf{q}}_{i-1}\mathbf{q}_i)$ that allow for multiplication by a scalar weight as needed for interpolation. With the cumulative basis $\tilde{N}$ and replacing $\mathbf{p}(\tau)$ with $\mathbf{q}(\tau)$, $\mathbf{p}_0$ with $\mathbf{q}_0$, $\mathbf{v}_i$ with $\boldsymbol{\omega}_i$, summation in the Lie algebra with (quaternion) multiplication in the Lie group ($\exp\sum_{i=1}^{n}\alpha_i = \prod_{i=1}^{n}\exp\alpha_i$) we arrive at:

$$\text{CuBsp}(\mathbf{q}(\tau)) = \mathbf{q}_0 \prod_{i=1}^{n} \exp\left(\tilde{N}_i^k(\tau)\log(\mathbf{q}_{i-1}^{-1}\mathbf{q}_i)\right) \tag{17}$$

When using uniform time intervals $\Delta\tau = \tau_{i+1} - \tau_i \forall i \in [0, n-1]$ the cubic cumulative B-spline basis matrix $\widetilde{\mathbf{N}}$ (18) for a $\mathcal{C}^2$ curve (Fig. 4d) is obtained by summing up the columns of the standard basis form matrix (2) with equal or greater index for each column [27]:

$$\begin{aligned}\widetilde{\mathbf{N}} &= [\,u^3 \quad u^2 \quad u \quad 1]\mathbf{C}\\ \dot{\widetilde{\mathbf{N}}} &= \frac{[3u^2 \quad 2u \quad 1 \quad 0]}{\Delta\tau}\mathbf{C}\end{aligned} \qquad \mathbf{C} = \frac{1}{6}\begin{bmatrix}0 & 1 & -2 & 1\\ 0 & -3 & 3 & 0\\ 0 & 3 & 3 & 0\\ 6 & 5 & 1 & 0\end{bmatrix} \tag{18}$$

The angular velocity of this cubic quaternion curve $\mathbf{q}(\tau) = \mathbf{q}_0 \exp(\widetilde{\mathbf{N}}_1\boldsymbol{\omega}_1)\exp(\widetilde{\mathbf{N}}_2\boldsymbol{\omega}_2)\exp(\widetilde{\mathbf{N}}_3\boldsymbol{\omega}_3)$ (17), where basis matrix $\widetilde{\mathbf{N}}$, but not control quaternions $\mathbf{q}_i$, depend on time τ through u can be derived analytically, which in an optimization problem allows to directly constrain the

trajectory shape using IMU measurements *e.g.* with automatic differentiation in Ceres Solver [1]. The angular velocity $\boldsymbol{\omega}(\tau) \in \mathbb{R}^3$ measured in $\mathbb{SO}(3)$ in the sensor frame is the vector part of the pure quaternion $[0, \boldsymbol{\omega}(\tau)] = 2(\bar{\mathbf{q}}(\tau)\dot{\mathbf{q}}(\tau))$ obtained via the first curve derivative:

$$\dot{\mathbf{q}}(\tau) = \mathbf{q}_0 x_1 \dot{\chi}_1 x_2 x_3 + \mathbf{q}_0 x_1 x_2 \dot{\chi}_2 x_3 + \mathbf{q}_0 x_1 x_2 x_3 \dot{\chi}_3 \quad (19)$$

analytically derived by applying:

1. *Power rule* $(x^n)' = nx^{n-1}$ to matrix $\widetilde{\mathbf{N}}$ to get $\dot{\widetilde{\mathbf{N}}}$ (18).
2. *Chain rule* $x_i' = (u_i(v_i))' = u_i'(v_i)v_i'$ with $\boldsymbol{\omega}_i := \log(\bar{\mathbf{q}}_{i-1}\mathbf{q}_i)$, $u_i = \exp(v_i)$ and $v_i = \widetilde{\mathbf{N}}_i\boldsymbol{\omega}_i$ to get $x_i' = \exp(\widetilde{\mathbf{N}}_i\boldsymbol{\omega}_i)(\dot{\widetilde{\mathbf{N}}}_i\boldsymbol{\omega}_i)$.
3. *Product rule* $(x_1x_2x_3)' = x_1'x_2x_3 + x_1x_2'x_3 + x_1x_2x_3'$ with $x_i := \exp(\widetilde{\mathbf{N}}_i\boldsymbol{\omega}_i)$, $\dot{\chi}_i := (\dot{\widetilde{\mathbf{N}}}_i\boldsymbol{\omega}_i)$ to get (19).

5. Rigid body motion interpolation methods

In addition to orientation, a rigid body motion additionally consists of a translation part $\mathbf{t} \in \mathbb{R}^3$ that also needs to be interpolated, either independently in Euclidean space (Sec. 3) in a split interpolation scheme, or jointly. For the latter, we will now introduce two different possible parameterizations that are geometrically equivalent but have a different computational complexity.

Special Euclidean group of rigid body motion ($\mathbb{SE}(3)$)

$$\mathbb{SE}(3) = \left\{ T = \begin{bmatrix} \mathrm{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \;\middle|\; \mathrm{R} \in \mathbb{SO}(3), \mathbf{t} \in \mathbb{R}^3 \right\} \quad (20)$$

To jointly parameterize the full rigid body motion, consisting of translation and rotation, homogeneous 4×4 matrices are usually used. Transformation **composition** is given by matrix multiplication TT' and the **inverse** transformation T^{-1} is just:

$$\mathrm{TT}' = \begin{bmatrix} \mathrm{RR}' & \mathrm{R}\mathbf{t}' + \mathbf{t} \\ 0 & 1 \end{bmatrix}, \; \mathrm{T}^{-1} = \begin{bmatrix} \mathrm{R}^T & -\mathrm{R}^T\mathbf{t} \\ 0 & 1 \end{bmatrix} \quad (21)$$

A vector $\mathbf{p} \in \mathbb{R}^3$ is **rotated and translated** by interpreting it as a column vector $\tilde{\mathbf{p}} = [p_x, p_y, p_z, 1]^T$ in homogeneous coordinates and then carrying out a simple matrix-vector multiplication: $\mathrm{T}\mathbf{p} := \mathrm{T}\tilde{\mathbf{p}} = \mathrm{R}\mathbf{p} + \mathbf{t}$.

Lie algebra of twists ($\mathfrak{se}(3)$)

$$\mathfrak{se}(3) = \left\{ \hat{\xi} = \begin{bmatrix} [\boldsymbol{\omega}]_\times & \mathbf{v} \\ 0 & 0 \end{bmatrix} \;\middle|\; [\boldsymbol{\omega}]_\times \in \mathfrak{so}(3), \mathbf{v} \in \mathbb{R}^3 \right\} \quad (22)$$

Being a Lie group, $\mathbb{SE}(3)$ also has its associated Lie algebra of twists $\hat{\xi} \in \mathfrak{se}(3)$ where the vectors $\boldsymbol{\omega}$ and $\mathbf{v}$ can be interpreted as angular and linear velocity. There also exist versions of $\mathbb{SE}(3)$ exponentiation and logarithm [23] to convert between algebra and group.

Dual quaternions ($\mathbb{DH}$) A *dual quaternion* $\mathbf{Q} \in \mathbb{DH}$ is an ordered set of quaternions $\mathbf{q}, \mathbf{q}'$ that can be written as $\mathbf{Q} = \mathbf{q} + \mathbf{q}'\varepsilon$ where ε is a *dual unit* satisfying $\varepsilon^2 = 0$ and commuting with the quaternion imaginary units, *e.g.* $i\varepsilon = \varepsilon i$ [15]. Its algebra is a combination of quaternion and dual number algebra, which mostly differs in the definition of the norm since there are two conjugates [19].

$$\mathbb{DH}_1 = \left\{ \mathbf{Q} = \mathbf{q} + \frac{1}{2}\tilde{\mathbf{t}}\mathbf{q}\varepsilon \;\middle|\; \tilde{\mathbf{t}} \in \mathbb{H}_0, \mathbf{q} \in \mathbb{H}_1 \right\} \quad (23)$$

A **unit** dual quaternion is an alternative and more space-efficient parameterization of the joint rigid body motion than transformation matrices. It has unit norm $||\mathbf{Q}|| = 1$, which translates to satisfying the constraints $\mathbf{q}\bar{\mathbf{q}} = 1$ and $\mathbf{q}\bar{\mathbf{q}}' + \mathbf{q}'\bar{\mathbf{q}} = 0$ [3]. They will be automatically fulfilled if it is constructed by interpreting the translation part $\mathbf{t}$ of the rigid body motion as a pure quaternion $\tilde{\mathbf{t}} = [0, \mathbf{t}]^T$ and the rotation part $\mathbf{q}$ as a unit quaternion. The translational part can be recovered via $[0, \mathbf{t}]^T = \tilde{\mathbf{t}} = 2\mathbf{q}'\bar{\mathbf{q}}$ [19, 15].

$$(\text{angle } \theta, \text{pitch } d, \text{axis } \mathbf{l}, \text{moment } \mathbf{m}) \in \mathbb{R}^8 \quad (24)$$

The transformation of unit dual quaternion parameters $(\mathbf{q}, \mathbf{q}') \in \mathbb{R}^8$ to **screw** parameters (24) is a way to represent their tangent space, which is geometrically equal to $\mathfrak{se}(3)$, but in which interpolation can be done just by scaling θ and d [5]. This also allows to implement the exp and log operations and thus the power of a unit dual quaternion more efficiently than the corresponding operations needed to get from transformation matrices $\mathrm{T} \in \mathbb{SE}(3)$ to twists $\hat{\xi} \in \mathfrak{se}(3)$ and twist coordinates $\xi \in \mathbb{R}^6$ [20].

5.1. Interpolating two keyframes (Fig. 5)

Split interpolation (SPLIT) [21] in $\mathbb{R}^3 \times \mathbb{SU}(2)$ as usually employed in graphics separately interpolates position and orientation by applying SLERP (14) on $\mathbf{q}_0, \mathbf{q}_1$ and LERP (1) on $\mathbf{t}_0, \mathbf{t}_1$ along $u \in [0, 1]$:

$$\begin{aligned} &\mathrm{SPLIT}\big((\mathbf{q}_0, \mathbf{t}_0), (\mathbf{q}_1, \mathbf{t}_1), u\big) \\ &\quad = \big(\mathrm{SLERP}(\mathbf{q}_0, \mathbf{q}_1, u), \mathrm{LERP}(\mathbf{t}_0, \mathbf{t}_1, u)\big) \\ &\quad = \big(\mathbf{q}_0 \exp(u \log(\bar{\mathbf{q}}_0\mathbf{q}_1)), \mathbf{t}_0 + u(\mathbf{t}_1 - \mathbf{t}_0)\big) \end{aligned} \quad (25)$$

Special Euclidean twist upsampler (SE3Up) [25, 7] in $\mathbb{SE}(3)$ is a SLERP (14) like construction on $\mathfrak{se}(3)$ (5) twists in the tangent space of $\mathbb{SE}(3)$ (20), resulting in a joint interpolation scheme. The orientation curve is identical to (25), but the translation curve does not follow the shortest path between the two keyframes. Instead, the intermediate translations along $u \in [0, 1]$ depend on the orientation difference between keyframes, as inspection of the $\mathfrak{se}(3)$ tangents (27) for the translational part $\mathrm{R}_0^T(\mathbf{t}_1 - \mathbf{t}_0)$ reveals:

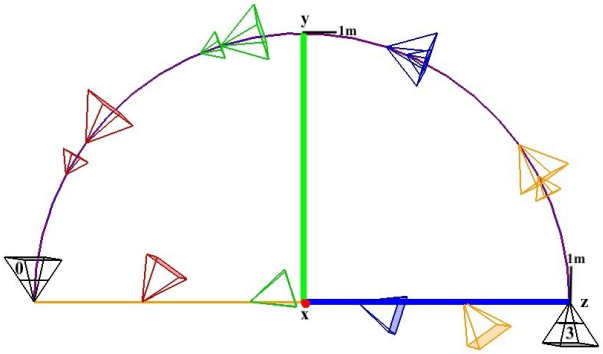


Figure 5: Interpolation of two poses (black) with positions $\mathbf{t}_0 = (0, 0, -1)$, $\mathbf{t}_3 = (0, 0, 1)$ and rotations with angle $\theta_0 = -90°, \theta_3 = 90°$ around the x axis in the y-z plane. The maximum translational deviation between SPLIT (orange) and SE3Up/ScLERP (violet) is half the distance of the two base poses. This happens when they are pointing in opposite directions, in which case joint interpolation follows a half-circle. DLB (small frustrums) traces out the same half-circle, but at non-constant (bell-shaped) speed resulting in the first 2 intermediate frames (red,green) beeing placed before, and the last two (blue,yellow) after their non-approximated counterpart.

$$\mathrm{SE3Up}(\mathrm{T}_0, \mathrm{T}_1, u) = \mathrm{T}_0 \exp\left(u \log(\mathrm{T}_0^{-1}\mathrm{T}_1)\right) \tag{26}$$

$$\log(\mathrm{T}_0^{-1}\mathrm{T}_1) = \log \begin{bmatrix} \mathrm{R}_0^T\mathrm{R}_1 & \mathrm{R}_0^T(\mathbf{t}_1 - \mathbf{t}_0) \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{27}$$

Screw linear interpolation (ScLERP) [15] in $\mathbb{DH}_1$ is geometrically equivalent but slightly more efficient than (26) because of the screw (24) tangent space parameterization. **Dual quaternion linear blending (DLB)** [15], the direct extension of QLB (9) to $\mathbb{DH}_1$, is a much more efficient approximation of (26) coming at the cost of non-constant angular and linear velocity between frames.

5.2. Trajectory from multiple keyframes (Fig. 6)

Split Trajectory (CuBsp) [21] in $\mathbb{R}^3 \times \mathbb{SU}(2)$ through iterated SPLIT (25) weighted as in (17) produces independent curves $(\mathbf{q}(\tau), \mathbf{t}(\tau))$. The cumulative basis $\widetilde{\mathbf{N}}$ (18) can with $\mathbf{v}_i = \mathbf{t}_i - \mathbf{t}_{i-1}$ also be used in $\mathbb{R}^3$ as $\mathbf{t}(\tau) = \mathbf{t}_0 + \widetilde{\mathbf{N}}_1\mathbf{v}_1 + \widetilde{\mathbf{N}}_2\mathbf{v}_2 + \widetilde{\mathbf{N}}_3\mathbf{v}_3$ instead of (2) directly on $\mathbf{t}_i$.

Spline Fusion twist curve (SpFus) [22, 27] in $\mathbb{SE}(3)$ through iterated SE3Up (26) leads to a joint trajectory whose translation $\mathbf{t}(\tau)$ is hard to control, since it is coupled with orientation differences between 4 control points.

Screw Fusion B-spline (ScFus) in $\mathbb{DH}_1$ through iterated ScLERP on screw (24) tangents, and its approximation **Dual quaternion linear fusion (DLFus)**, which is DLB weighted by (2) are faster than SpFus but behave similarly.

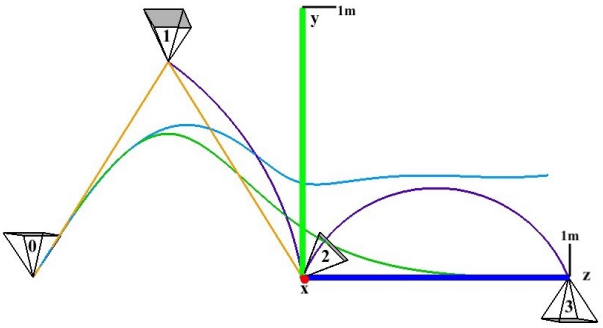


Figure 6: Trajectory from 4 poses with $\mathbf{t}_1 = (0, 0.8, -0.5)$, $\mathbf{t}_2 = (0, 0, 0)$, $\theta_1 = \theta_0$, $\theta_2 = -\pi/4$. The translational deviation between piecewise SPLIT (orange) and SE3Up (violet) declines in proportion to the relative orientation difference and the curves are equal when it is 0°. The translation of the higher order split trajectory CuBsp (green) is the weighted sum of the positions of its base poses, so it behaves just like a normal B-spline in Euclidean space, which is not the case for the joint trajectory SpFus/ScFus (cyan). For the cubic case as above, its shape additionally depends on the relative orientation differences of 4 base poses, weighted by (18).

6. Runtime experiments

Our experiments (Tab. 2) show that joint interpolation with twists in $\mathbb{SE}(3)$ (SE3Up, SpFus) is 2x slower than the split one in $\mathbb{R}^3 \times \mathbb{SU}(2)$ (SPLIT, CuBsp), which is consistent with [25], who reported that an optimization algorithm using CuBsp converges twice as fast as SpFus. Joint interpolation with screws in $\mathbb{DH}_1$ (ScLERP, ScFus) is just 1.3x slower. Approximations (QLB+LERP, DLB, DLFus) are 10x, 5x, at least 2x faster than exact ones (SPLIT, CuBsp).

Name	Time(ns)	Name	Time(ns)
QLB+LERP	10	DLFus	167
DLB	23	SQUAD*	189
SPLIT	103	CuBsp	319
ScLERP	127	ScFus	428
SE3Up	200	SpFus	675

Table 2: Runtime for pairwise (left) and higher order (right) rigid body motion interpolation. Timings are the averages for one evaluation of the curve on an i7-8700K@3.7GHz with Ubuntu 18.04, g++ 7.3.0, Eigen 3.3.4 [11] and Sophus #13fb3288 [31]. (*) For the translation part, we applied a similar bilinear interpolation as in SQUAD by replacing SLERP with LERP in (15).

7. Conclusion

We compared Euclidean, orientation and rigid body motion interpolation methods on $\mathbb{SU}(2)$, $\mathbb{SE}(3)$ and $\mathbb{DH}_1$ manifolds. Splitting $\mathbb{R}^3 \times \mathbb{SU}(2)$ (CuBsp) avoids coupling translation with orientation as in the joint trajectory in $\mathbb{SE}(3)$ (SpFus), efficiently represented via $\mathbb{DH}_1$ (ScFus/DLFus). In the future, higher spline orders and the methods' convergence rates in optimization procedures will be evaluated.

References

[1] S. Agarwal, K. Mierle, and Others. Ceres solver. http://ceres-solver.org, 2012. 7

[2] B. Busam, T. Birdal, and N. Navab. Camera pose filtering with local regression geodesics on the riemannian manifold of dual quaternions. October 2017. 2

[3] B. Busam, M. Esposito, B. Frisch, and N. Navab. Quaternionic upsampling: Hyperspherical techniques for 6 dof pose tracking. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 629–638. IEEE, 2016. 2, 7

[4] E. B. Dam, M. Koch, and M. Lillholm. *Quaternions, interpolation and animation*. Datalogisk Institut, Københavns Universitet, 1998. 2, 4, 6

[5] K. Daniilidis. Hand-eye calibration using dual quaternions. *The International Journal of Robotics Research*, 18(3):286–298, 1999. 2, 7

[6] G. Farin. *Curves and Surfaces for CAGD*. 5 edition, 2002. 3

[7] P.-E. Forssén. Smoothing of so(3) and se(3). slerp, and splines on so(3). Geometry for Computer Vision PhD course, Slides for lecture 7, 2014. 2, 7

[8] P. Furgale, T. D. Barfoot, and G. Sibley. Continuous-time batch estimation using temporal basis functions. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2088–2095. IEEE, 2012. 2

[9] P. Furgale, C. H. Tong, T. D. Barfoot, and G. Sibley. Continuous-time batch trajectory estimation using temporal basis functions. *The International Journal of Robotics Research*, page 0278364915585860, 2015. 2

[10] F. S. Grassia. Practical parameterization of rotations using the exponential map. *Journal of graphics tools*, 3(3):29–48, 1998. 1

[11] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010. 8

[12] A. J. Hanson. *Visualizing Quaternions*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006. 2

[13] D. D. Holm. *Geometric mechanics*. Imperial College Press, 2008. 5

[14] Y.-B. Jia. Quaternions and rotations. *Com S*, 477(577):15, 2008. 4

[15] L. Kavan, S. Collins, C. O'Sullivan, and J. Zara. Dual quaternions for rigid transformation blending. *Trinity College Dublin, Tech. Rep. TCD-CS-2006-46*, 2006. 2, 7, 8

[16] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan. Skinning with dual quaternions. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 39–46. ACM, 2007. 2

[17] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM Transactions on Graphics (TOG)*, 27(4):105, 2008. 2

[18] L. Kavan and J. Žára. Spherical blend skinning: a real-time deformation of articulated models. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 9–16. ACM, 2005. 2, 5

[19] B. Kenwright. A beginners guide to dual-quaternions: what they are, how they work, and how to use them for 3d character hierarchies. 2012. 7

[20] B. Kenwright. Dual-quaternions, from classical mechanics to computer graphics and beyond, 2012. 7

[21] 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. ACM, 1995. 2, 5, 6, 7, 8

[22] S. Lovegrove, A. Patron-Perez, and G. Sibley. Spline fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras. In *BMVC*, 2013. 2, 5, 8

[23] Y. Ma, S. Soatto, J. Kosecká, and S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Interdisciplinary Applied Mathematics. Springer New York, 2005. 5, 7

[24] M. E. Mortenson. *Geometric Modeling (2nd Ed.)*. John Wiley & Sons, Inc., New York, NY, USA, 2 edition, 1997. 3

[25] H. Ovrén and P.-E. Forssén. Trajectory representation and landmark projection for continuous-time structure from motion. *arXiv preprint arXiv:1805.02543*, 2018. 2, 7, 8

[26] R. Parent. *Computer animation: algorithms and techniques*. Newnes, 3 edition, 2012. 1, 3

[27] A. Patron-Perez, S. Lovegrove, and G. Sibley. A spline-based trajectory representation for sensor fusion and rolling shutter cameras. *International Journal of Computer Vision*, 113(3):208–219, 2015. 2, 5, 6, 8

[28] K. Shoemake. Animating rotation with quaternion curves. In *ACM SIGGRAPH computer graphics*, volume 19, pages 245–254. ACM, 1985. 2, 5, 6

[29] K. Shoemake. Quaternion calculus and fast animation. In *ACM SIGGRAPH Course Notes 10, Computer Animation: 3-D motion specification and control*, number 10, pages 101–121. Siggraph, 1987. 2, 5, 6

[30] H. Sommer, J. R. Forbes, R. Siegwart, and P. Furgale. Continuous-time estimation of attitude using b-splines on lie groups. *Journal of Guidance, Control, and Dynamics*, 39(2):242–261, 2015. 2

[31] H. Strasdat, S. Lovegrove, and Others. Sophus: C++ implementation of lie groups using eigen. https://github.com/strasdat/Sophus, 2011. 8

[32] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques*. ACM, New York, NY, USA, 1991. 6