# Efficient Derivative Computation for Cumulative B-Splines on Lie Groups

Christiane Sommer* Vladyslav Usenko* David Schubert Nikolaus Demmel Daniel Cremers
Technical University of Munich

## Abstract

*Continuous-time trajectory representation has recently gained popularity for tasks where the fusion of high-frame-rate sensors and multiple unsynchronized devices is required. Lie group cumulative B-splines are a popular way of representing continuous trajectories without singularities. They have been used in near real-time SLAM and odometry systems with IMU, LiDAR, regular, RGB-D and event cameras, as well as for offline calibration.*

*These applications require efficient computation of time derivatives (velocity, acceleration), but all prior works rely on a computationally suboptimal formulation. In this work we present an alternative derivation of time derivatives based on recurrence relations that needs $\mathcal{O}(k)$ instead of $\mathcal{O}(k^2)$ matrix operations (for a spline of order $k$) and results in simple and elegant expressions. While producing the same result, the proposed approach significantly speeds up the trajectory optimization and allows for computing simple analytic derivatives with respect to spline knots. The results presented in this paper pave the way for incorporating continuous-time trajectory representations into more applications where real-time performance is required.*

## 1. Introduction

Estimating trajectories is a recurring topic in computer vision research: In odometry and SLAM applications the sensor motion needs to be estimated, in object tracking and robotic grasping tasks, we want to compute the 6DoF pose over time, and for autonomous exploration, path planning and obstacle avoidance, we need to predict good trajectories. Over the last years, researchers have increasingly reverted to *continuous-time trajectories*: Instead of a simple list of poses for discrete time points, the trajectory is elegantly represented by a continuous function in time with values in the space of possible poses. B-splines are a natural choice for parameterizing such functions. They have been used in several well-known works on continuous-time trajectory estimation. However, since the B-spline trajectories take values in the Lie group of poses, the resulting differential calculus is much more involved than in Euclidean space $\mathbb{R}^d$. Existing approaches to computing time derivatives suffer from a high computational cost that is actually quadratic in the spline order.

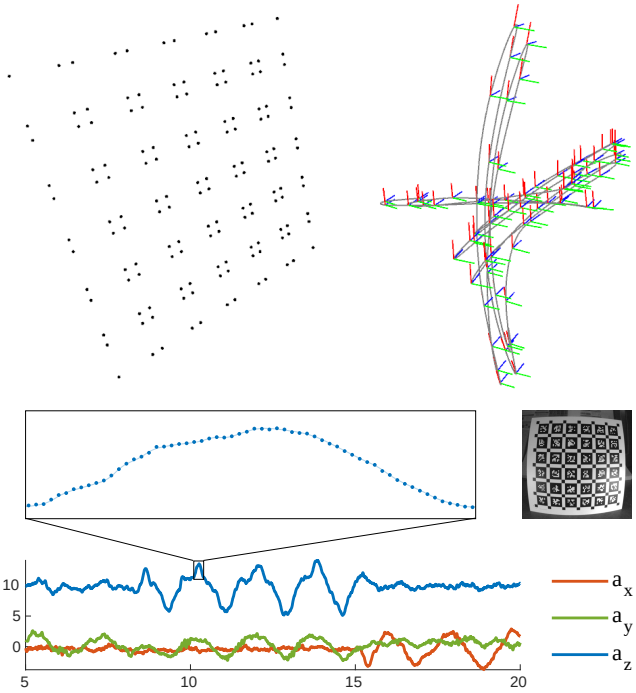


Figure 1. Camera-IMU calibration using a Lie group cumulative B-spline to represent the IMU trajectory (gray line with axes used to visualize rotation). Observations of the calibration pattern are combined with accelerometer and gyroscope measurements to estimate the trajectory and calibration parameters in a joint optimization. The plot visualizes the accelerometer measurements in m/s$^2$ (dots) overlaid on the continuous estimate generated from the spline trajectory (line) after optimization. As shown in the experimental section, the proposed formulation is able to significantly reduce the computational effort of such an optimization.

In this paper, we introduce recurrence relations for respective time derivatives and show how they can be employed to significantly reduce the computational cost and to derive concrete (analytic) expressions for the spline Jacobians w.r.t. the control points. This is not only of theoret-

* These authors contributed equally.

ical interest: by speeding up time derivatives and Jacobian computation significantly, we take a large step towards the real-time capability of continuous-time trajectory representation and its applications such as camera tracking, motion planning, object tracking or rolling-shutter modelling. In summary, our contributions are the following:

- A simple formulation for the time derivatives of Lie group cumulative B-splines that requires a number of matrix operation which scales linearly with the order $k$ of the spline.
- Simple (linear in $k$) analytic Jacobians of the value and the time derivatives of an $SO(3)$ spline with respect to its knots.
- Faster optimization time compared to the currently available implementations, due to provably lower complexity. This is demonstrated on simulated experiments and real-world applications such as camera-IMU calibration.

## 2. Related Work

This paper consists of two main parts: first, we take a detailed look at the theory behind B-splines, in particular on Lie groups. In the second part, we look at possible applications of our efficient derivative computation in computer vision. In the following, we will review related work for both parts.

**B-splines in Lie groups** Since the 1950s, B-splines have become a popular tool for approximating and interpolating functions of one variable. Most notably, de Casteljau [6], Cox [4] and De Boor [5] introduced principled ways of deriving spline coefficients from a set of desirable properties, such as locality and smoothness. Qin [17] found that due to their locality property, B-splines are conveniently expressed using a matrix representation. By using so-called cumulative B-splines, the concept can be transferred from $\mathbb{R}^d$-valued functions to the more general set of Lie group-valued functions. This was first done for the group of 3D rotations $SO(3)$ [12], and later generalized to arbitrary Lie groups $\mathcal{L}$ [21]. The latter also contains formulas for computing derivatives of $\mathcal{L}$-valued B-splines, but the way they are formulated is not practical for implementation. For a general overview of computations in Lie groups and Lie algebras, we refer to [2, 20].

**Applications in computer vision** Thanks to their flexibility in representing functions, B-splines have been used a lot for trajectory representation in computer vision and robotics. The applications range from calibration [8, 13] to odometry estimation with different sensors [13, 11, 14], 3D reconstruction [15, 16] and trajectory planning [22, 7]. All of these works need temporal derivatives of the B-splines at some point, but to the best of our knowledge, there is no work explicitly investigating efficient computation and complexity of these. Several works have addressed the question if it is better to represent trajectories as one spline in $SE(3)$, or rather use a split representation of two splines in $\mathbb{R}^3$ and $SO(3)$ [9, 15, 16]. While this cannot be answered unambiguously without looking at the specific use case, all authors come to the conclusion that on average, using the split representation is better both in terms of trajectory representation and in terms of computation time.

## 3. Lie Group Foundations

### 3.1. Notation

A Lie group $\mathcal{L}$ is a group which also is a differentiable manifold, and for which group multiplication and inversion are differentiable operations. The corresponding Lie algebra $\mathcal{A}$ is the tangent space of $\mathcal{L}$ at the identity element $\mathbb{1}$. Prominent examples of Lie groups are the trivial vector space Lie groups $\mathbb{R}^d$, which have $\mathcal{L} = \mathcal{A} = \mathbb{R}^d$ and where the group multiplication is simple vector addition, and matrix Lie groups such as the transformation groups $SO(n)$ and $SE(n)$, with matrix multiplication as group multiplication. Of particular interest in computer vision applications are the groups $SO(3)$ of 3D rotations and $SE(3)$, the group of rigid body motions.

The continuous-time trajectories in this paper are functions of time $t$ with values in a Lie group $\mathcal{L}$. If $d$ denotes the number of degrees of freedom of $\mathcal{L}$, the hat transform $\cdot_\wedge \colon \mathbb{R}^d \to \mathcal{A}$ is used to map tangent vectors to elements in the Lie algebra $\mathcal{A}$. The Lie algebra elements can be mapped to their Lie group elements using the matrix exponential $\exp \colon \mathcal{A} \to \mathcal{L}$, which has a closed-form expression for $SO(3)$ and $SE(3)$. The composition of the hat transform followed by the matrix exponential is given by

$$\operatorname{Exp} \colon \mathbb{R}^d \to \mathcal{L}\,. \tag{1}$$

Its inverse is denoted

$$\operatorname{Log} \colon \mathcal{L} \to \mathbb{R}^d\,, \tag{2}$$

which is a composition of the matrix logarithm $\log \colon \mathcal{L} \to \mathcal{A}$ followed be the inverse of the hat transform $\cdot_\vee \colon \mathcal{A} \to \mathbb{R}^d$.

**Definition 3.1.** *For an element $R \in \mathcal{L}$, the* adjoint $\operatorname{Adj}_R$ *is the linear map defined by*

$$\operatorname{Adj}_R \boldsymbol{\xi} = (R \boldsymbol{\xi}_\wedge R^{-1})_\vee \qquad \textit{for} \quad \boldsymbol{\xi} \in \mathbb{R}^d\,. \tag{3}$$

It follows readily from the definition in (3) that the following relations hold true for any $\boldsymbol{\xi} \in \mathbb{R}^d$:

$$R \operatorname{Exp}(\boldsymbol{\xi}) = \operatorname{Exp}(\operatorname{Adj}_R \boldsymbol{\xi}) R\,, \tag{4}$$

$$\operatorname{Exp}(\boldsymbol{\xi}) R = R \operatorname{Exp}(\operatorname{Adj}_{R^{-1}} \boldsymbol{\xi})\,. \tag{5}$$

If $R \in SO(3)$, the adjoint is simply $\mathrm{Adj}_R = R$. In this paper, we also use the commutator of two Lie algebra elements:

**Definition 3.2.** *The* commutator *is defined as*

$$[\cdot,\cdot] : \mathcal{A} \times \mathcal{A} \to \mathcal{A}, \quad [A, B] = AB - BA\,. \tag{6}$$

For $A = \mathbf{a}_\wedge, B = \mathbf{b}_\wedge \in \mathfrak{so}(3)$, the commutator has the property

$$[A, B]_\vee = \mathbf{a} \times \mathbf{b}\,. \tag{7}$$

### 3.2. Differentiation

To differentiate the trajectories with respect to their parameters, the following definitions and conventions will be used.

**Definition 3.3.** *The* right Jacobian $J_r$ *is defined by*

$$J_r(\boldsymbol{\xi})\mathbf{v} = \lim_{\epsilon \to 0} \frac{\mathrm{Log}(\mathrm{Exp}(\boldsymbol{\xi})^{-1}\,\mathrm{Exp}(\boldsymbol{\xi} + \epsilon\mathbf{v}))}{\epsilon} \tag{8}$$

*for all vectors* $\mathbf{v} \in \mathbb{R}^d$.

Intuitively, the right Jacobian measures how the difference of $\mathrm{Exp}(\boldsymbol{\xi})$ and $\mathrm{Exp}(\boldsymbol{\xi} + \mathbf{v})$, mapped back to $\mathbb{R}^d$, changes with $\mathbf{v}$. It has the following properties:

$$\mathrm{Log}(\mathrm{Exp}(\boldsymbol{\xi})\,\mathrm{Exp}(\boldsymbol{\delta})) = \boldsymbol{\xi} + J_\mathrm{r}(\boldsymbol{\xi})^{-1}\boldsymbol{\delta} + \mathcal{O}(\boldsymbol{\delta}^2)\,, \tag{9}$$

$$\mathrm{Exp}(\boldsymbol{\xi} + \boldsymbol{\delta}) = \mathrm{Exp}(\boldsymbol{\xi})\,\mathrm{Exp}(J_\mathrm{r}(\boldsymbol{\xi})\boldsymbol{\delta}) + \mathcal{O}(\boldsymbol{\delta}^2)\,. \tag{10}$$

If $\mathcal{L} = SO(3)$, the right Jacobian and its inverse can be found in [3, p. 40].

Whenever an expression $f(R)$ is differentiated w.r.t. to a Lie group element $R$, the derivative is defined as

$$\frac{\partial f(R)}{\partial R} = \left.\frac{\partial f(\mathrm{Exp}(\boldsymbol{\delta})R)}{\partial \boldsymbol{\delta}}\right|_{\boldsymbol{\delta}=0}. \tag{11}$$

Consequently, an update step for the variable $R$ during optimization is performed as $R \leftarrow \mathrm{Exp}(\boldsymbol{\delta})R$, where $\boldsymbol{\delta}$ is the the increment determined by the optimization algorithm.

## 4. B-Spline Foundations

### 4.1. Basics

B-splines define a continuous function using a set of control points (knots), see also Fig. 2. They have a number of desirable properties for continuous trajectory representation, in particular locality and $C^{k-1}$ smoothness for a spline of order $k$ (degree $k-1$). We will focus on uniform B-splines of order $k$ in this work.

**Definition 4.1.** *A* uniform B-spline *of order* $k$ *is defined by its control points* $\mathbf{p}_i$ *at times* $t_i = t_0 + i\Delta t$ $(0 \le i \le N)$ *and a set of spline coefficients* $B_{i,k}(t)$*:*

$$\mathbf{p}(t) = \sum_{i=0}^{N} B_{i,k}(t)\,\mathbf{p}_i\,, \tag{12}$$

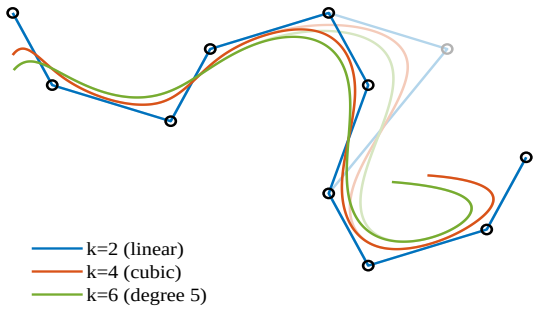


Figure 2. A set of control points (black) in $\mathbb{R}^2$, and the resulting B-splines for different orders $k$. For the linear spline ($k = 2$), the spline curve actually hits the control points, while for higher order splines, this is not true in general. The lighter lines show how the splines change if one control point changes: the curves only change locally, i.e. in the vicinity of the modified control point.

*where the coefficients are given by the De Boor–Cox recurrence relation [4, 5]*

$$B_{i,0}(t) = \begin{cases} 1, & \textit{for } t_i \le t < t_{i+1} \\ 0, & \textit{otherwise} \end{cases} \tag{13}$$

$$B_{i,j}(t) = \frac{t - t_i}{j\Delta t} B_{i,j-1}(t) + \frac{t_{i+j+1} - t}{j\Delta t} B_{i+1,j-1}(t)\,. \tag{14}$$

It is possible to transform (12) into a cumulative representation:

$$\mathbf{p}(t) = \widetilde{B}_{0,k}(t)\,\mathbf{p}_0 + \sum_{i=1}^{N} \widetilde{B}_{i,k}(t)(\mathbf{p}_i - \mathbf{p}_{i-1})\,, \tag{15}$$

$$\widetilde{B}_{i,k}(t) = \sum_{s=i}^{N} B_{s,k}(t)\,. \tag{16}$$

### 4.2. Matrix Representation

B-splines have local support, which means that for a spline of order $k$, only $k$ control points contribute to the value of the spline. As shown in [17], it is possible to represent the spline coefficients using a matrix representation, which is constant for uniform B-splines.

At time $t \in [t_i, t_{i+1})$ the value of $\mathbf{p}(t)$ only depends on the control points $t_i$, $t_{i+1}$, ..., $t_{i+k-1}$. To simplify calculations, we transform time to a uniform representation $s(t) := (t - t_0)/\Delta t$, such that the control points transform into $\{0, .., k-1\}$. We define $u(t) := s(t) - i$ as normalized time elapsed since the start of the segment $[t_i, t_{i+1})$ and from now on use $u$ as temporal variable. The value of $\mathbf{p}(u)$ can then be evaluated using a matrix representation as follows [17]:

$$\mathbf{p}(u) = \begin{pmatrix} \mathbf{p}_i, & \mathbf{p}_{i+1}, & \cdots, & \mathbf{p}_{i+k-1} \end{pmatrix} M^{(k)}\mathbf{u}\,, \tag{17}$$

where $u_n = u^n$, $M^{(k)}$ is a blending matrix with entries

$$m_{s,n}^{(k)} = \frac{C_{k-1}^n}{(k-1)!} \sum_{l=s}^{k-1} (-1)^{l-s} C_k^{l-s} (k-1-l)^{k-1-n}, \quad s,n \in \{0,\dots,k-1\}, \tag{18}$$

and $C_k^s = \frac{k!}{s!\,(k-s)!}$ are binomial coefficients.

It is also possible to use the matrix representation for the cumulative splines:

$$\mathbf{p}(u) = \begin{pmatrix} \mathbf{p}_i, & \mathbf{d}_1^i, & \cdots, & \mathbf{d}_{k-1}^i \end{pmatrix} \widetilde{M}^{(k)} \mathbf{u}, \tag{19}$$

with cumulative matrix entries $\widetilde{m}_{j,n}^{(k)} = \sum_{s=j}^{k-1} m_{s,n}^{(k)}$ and difference vectors $\mathbf{d}_j^i = \mathbf{p}_{i+j} - \mathbf{p}_{i+j-1}$.

We show in the Appendix that the first row of the matrix $\widetilde{M}^{(k)}$ is always equal to the unit vector $\mathbf{e}_0 \in \mathbb{R}^k$:

$$\widetilde{m}_{0,n}^{(k)} = \delta_{n,0}. \tag{20}$$

In particular, if we define

$$\boldsymbol{\lambda}(u) = \widetilde{M}^{(k)} \mathbf{u}, \tag{21}$$

this implies $\lambda_0(u) \equiv 1$. Inserting $\boldsymbol{\lambda}$ with $\lambda_0 = 1$ into the cumulative matrix representation (19) allows us to write $\mathbf{p}(u)$ conveniently as follows:

**Theorem 4.2.** *The B-spline of order $k$ at position $u$ can be written as*

$$\mathbf{p}(u) = \mathbf{p}_i + \sum_{j=1}^{k-1} \lambda_j(u) \cdot \mathbf{d}_j^i. \tag{22}$$

## 5. Cumulative B-splines in Lie groups

The cumulative B-spline in (22) can be generalized to Lie groups [21], and in particular to $SO(3)$ for smooth rotation generation [12]. First, a simple $\mathbb{R}^d$-addition in (22) corresponds to the group multiplication in a general Lie group (matrix multiplication in matrix Lie groups). Second, while we can easily scale a vector $\mathbf{d} \in \mathbb{R}^d$ by a factor $\lambda \in \mathbb{R}$ using scalar multiplication in $\mathbb{R}^d$, the concept of scaling does not exist for elements $R$ of a Lie group. Thus, we first have to map $R$ from $\mathcal{L}$ to the Lie algebra $\mathcal{A}$, which is a vector space, then scale the result, and finally map it back to $\mathcal{L}$: $\mathrm{Exp}(\lambda \cdot \mathrm{Log}(R))$. These two observations together lead to the following definition of cumulative B-splines in Lie groups:

**Definition 5.1.** *The cumulative B-spline of order $k$ in a Lie group $\mathcal{L}$ with control points $R_0, \cdots, R_N \in \mathcal{L}$ has the form*

$$R(u) = R_i \cdot \prod_{j=1}^{k-1} \mathrm{Exp}\left(\lambda_j(u) \cdot \mathbf{d}_j^i\right), \tag{23}$$

*with the generalized difference vector $\mathbf{d}_j^i$*

$$\mathbf{d}_j^i = \mathrm{Log}\left(R_{i+j-1}^{-1} R_{i+j}\right) \in \mathbb{R}^d. \tag{24}$$

We should mention that as opposed to a B-spline in $\mathbb{R}^d$, the order of multiplication (addition in (22)) does matter here, and different generalizations to Lie groups are possible in principle. In practice, we use the convention that is most commonly used in related work [13, 11, 14, 15, 16]. We omit the $i$ to simplify notation, and define

$$A_j(u) = \mathrm{Exp}\left(\lambda_j(u) \cdot \mathbf{d}_j\right) \tag{25}$$

to obtain the more concise expression

$$R(u) = R_i \cdot \prod_{j=1}^{k-1} A_j(u). \tag{26}$$

Note that this can be re-formulated as a recurrence relation:

$$R(u) = R^{(k)}(u), \tag{27}$$

$$R^{(j)}(u) = R^{(j-1)}(u) A_{j-1}(u), \tag{28}$$

$$R^{(1)}(u) = R_i. \tag{29}$$

### 5.1. Time derivatives

The main contribution of this paper is a simplified representation of the temporal derivatives compared to related work, which needs less operations. We first review the derivatives according to the current standard, and then introduce ours. We denote differentiation w.r.t. $u$ by a dot and apply the product rule to get

$$\dot{R}(u) = R_i \cdot \sum_{j=1}^{k-1} \left( \prod_{l=1}^{j-1} A_l(u) \right) \dot{A}_j(u) \left( \prod_{l=j+1}^{k-1} A_l(u) \right) \tag{30}$$

with

$$\dot{A}_j(u) = \dot{\lambda}_j(u) A_j(u) D_j = \dot{\lambda}_j(u) D_j A_j(u), \tag{31}$$

and $D_j = (\mathbf{d}_j)_\wedge$. Note that $A_j(u)$ and $D_j$ commute by definition. For the case of cubic splines ($k = 4$), this reduces to

$$\dot{R} = R_i \left( \dot{A}_1 A_2 A_3 + A_1 \dot{A}_2 A_3 + A_1 A_2 \dot{A}_3 \right), \tag{32}$$

which is the well known formula from e.g. [13, 14, 15, 16]. An implementation following this formula needs to perform $(k-1)^2 + 1$ matrix-matrix multiplications and is thus quadratic in the spline degree. We propose to define the time derivatives recursively instead, which needs less operations:

**Theorem 5.2.** *The time derivative $\dot{R}$ is given by the following recurrence relation:*

$$\dot{R} = R \boldsymbol{\omega}_\wedge^{(k)}, \tag{33}$$

$$\boldsymbol{\omega}^{(j)} = \mathrm{Adj}_{A_{j-1}^{-1}} \boldsymbol{\omega}^{(j-1)} + \dot{\lambda}_{j-1} \mathbf{d}_{j-1} \in \mathbb{R}^d, \tag{34}$$

$$\boldsymbol{\omega}^{(1)} = \mathbf{0} \in \mathbb{R}^d. \tag{35}$$

*$\boldsymbol{\omega}^{(k)}$ is commonly referred to as* velocity. *For $\mathcal{L} = SO(n)$, we also call it* rotational velocity.

*Proof.* We use the recursive definition of $R(u)$ in (28) and prove by induction over $j$ that

$$\dot{R}^{(j)} = R^{(j)} \boldsymbol{\omega}_\wedge^{(j)} , \tag{36}$$

which is equivalent to the claim for $j = k$. First, we note that for $j = 1$, $R^{(j)}(u) = R_i$ is constant w.r.t. $u$, and thus $\dot{R}^{(1)} = 0 = R^{(1)} \hat{\boldsymbol{\omega}}_\wedge^{(1)}$. Now, let (36) be true for some $j \in \{1, \dots, k-1\}$, then we have

$$\begin{aligned} \dot{R}^{(j+1)} &= \partial_u \left( R^{(j)} A_j \right) = \dot{R}^{(j)} A_j + R^{(j)} \dot{A}_j \\ &= R^{(j)} \left( \boldsymbol{\omega}_\wedge^{(j)} A_j + \dot{\lambda}_k A_j D_j \right) \\ &= R^{(j)} A_j \left( A_j^{-1} \boldsymbol{\omega}_\wedge^{(j)} A_j + \dot{\lambda}_j (\mathbf{d}_j)_\wedge \right) \\ &= R^{(j+1)} \underbrace{\left( \left( \mathrm{Adj}_{A_j^{-1}} \boldsymbol{\omega}^{(j)} \right)_\wedge + \dot{\lambda}_j (\mathbf{d}_j)_\wedge \right)}_{= \boldsymbol{\omega}_\wedge^{(j+1)}} . \end{aligned} \tag{37}$$

□

Note that our recursive definition of $\dot{R}$ makes it very easy to see that $R^{-1}\dot{R} \in \mathcal{A}$ for any Lie group, a property which is implicitly used in many works [13, 16], but never shown explicitly for arbitrary $\mathcal{L}$.

The scheme presented in Theorem 5.2 computes time derivatives with only $k-1$ matrix-vector multiplications and vector additions, together with one single matrix-matrix multiplication.

Since the case $\mathcal{L} = SO(3)$ is a common and important use case of B-splines in Lie groups, we explicitly state (34) for $R \in SO(3)$:

$$\boldsymbol{\omega}^{(j)} = A_{j-1}^\top \boldsymbol{\omega}^{(j-1)} + \dot{\lambda}_{j-1} \mathbf{d}_{j-1} . \tag{38}$$

For second order time derivatives, it is easy to see that the calculations proposed in related works [13, 14] need

$$k(k-1) + k\, C_{k-1}^2 = \frac{1}{2} k^2 (k-1) \tag{39}$$

matrix-matrix multiplications and are thus cubic in the spline order. We propose a different way to compute $\ddot{R}$:

**Theorem 5.3.** *The second derivative of $R$ w.r.t. $u$ can be computed by the following recurrence relation:*

$$\ddot{R} = R \left( (\boldsymbol{\omega}^{(k)})_\wedge^2 + \dot{\boldsymbol{\omega}}_\wedge^{(k)} \right) , \tag{40}$$

*where the* (rotational) acceleration $\dot{\boldsymbol{\omega}}^{(k)}$ *is recursively defined by*

$$\begin{aligned} \dot{\boldsymbol{\omega}}^{(j)} &= \dot{\lambda}_{j-1} \left[ \boldsymbol{\omega}_\wedge^{(j)}, D_{j-1} \right]_\vee \\ &\quad + \mathrm{Adj}_{A_{j-1}^{-1}} \dot{\boldsymbol{\omega}}^{(j-1)} + \ddot{\lambda}_{j-1} \mathbf{d}_{j-1} , \end{aligned} \tag{41}$$

$$\dot{\boldsymbol{\omega}}^{(1)} = \mathbf{0} \in \mathbb{R}^d . \tag{42}$$

*Proof.* (40) follows from (33) and the product rule. For (41), the last summand is trivial, so we focus on the derivative of the first term in (34): first, we note that

$$\mathrm{Adj}_{A_{j-1}^{-1}} \boldsymbol{\omega}^{(j-1)} = \left( A_{j-1}^{-1} \boldsymbol{\omega}_\wedge^{(j-1)} A_{j-1} \right)_\vee =: \bar{\boldsymbol{\omega}} , \tag{43}$$

so the time derivative of that term consists of three terms following the product rule for differentiation. The middle term is

$$\left( A_{j-1}^{-1} \dot{\boldsymbol{\omega}}_\wedge^{(j-1)} A_{j-1} \right)_\vee = \mathrm{Adj}_{A_{j-1}^{-1}} \dot{\boldsymbol{\omega}}^{(j-1)} , \tag{44}$$

which is exactly the second summand in (41). The remaining two terms are

$$\begin{aligned} &\left( \dot{A}_{j-1}^{-1} \boldsymbol{\omega}_\wedge^{(j-1)} A_{j-1} + A_{j-1}^{-1} \boldsymbol{\omega}_\wedge^{(j-1)} \dot{A}_{j-1} \right)_\vee \\ &\overset{(31)}{=} \left( -\dot{\lambda}_{j-1} D_{j-1} \bar{\boldsymbol{\omega}}_\wedge + \bar{\boldsymbol{\omega}}_\wedge \dot{\lambda}_{j-1} D_{j-1} \right)_\vee \\ &\overset{(6)}{=} \dot{\lambda}_{j-1} \left[ \bar{\boldsymbol{\omega}}_\wedge, D_{j-1} \right]_\vee \\ &\overset{(34)}{=} \dot{\lambda}_{j-1} \left[ \boldsymbol{\omega}_\wedge^{(j)} - \dot{\lambda}_{j-1} D_{j-1}, D_{j-1} \right]_\vee \\ &= \dot{\lambda}_{j-1} \left[ \boldsymbol{\omega}_\wedge^{(j)}, D_{j-1} \right]_\vee . \end{aligned} \tag{45}$$

□

This proposed scheme computes second time derivatives with only $2k$ matrix-matrix multiplications, $k-1$ matrix-vector multiplications, $3(k-1)$ vector additions and one matrix addition in any $\mathcal{L}$. For $\mathcal{L} = SO(3)$, $\dot{\boldsymbol{\omega}}^{(j)}$ in (41) simplifies to

$$\dot{\lambda}_{j-1} \boldsymbol{\omega}^{(j)} \times \mathbf{d}_{j-1} + A_{j-1}^\top \dot{\boldsymbol{\omega}}^{(j-1)} + \ddot{\lambda}_{j-1} \mathbf{d}_{j-1} . \tag{46}$$

This implies that for $SO(3)$, second order time derivatives only need $3(k-1)$ matrix-vector multiplications and vector additions plus two matrix-matrix multiplications, reducing computation time even further.

The iterative scheme for the computation of time derivatives can be extended to higher order derivatives. As an example, we provide third order time derivatives of $R$ together with the jerk $\dddot{\omega}^{(k)}$ in the Appendix. The number of matrix operations needed to compute this is still linear in the order of the spline. We also provide a comprehensive overview of the matrix operations needed for the different approaches in the Appendix.

### 5.2. Jacobians w.r.t. control points in $SO(3)$

The values of both the spline itself and its velocity and acceleration depends on the choice of control points. For the derivatives w.r.t. the control points of the B-spline, we first note that a control point $R_{i+j}$ appears implicitly in $\mathbf{d}_j$ and $\mathbf{d}_{j+1}$, and for $j = 0$, we also have the explicit dependence of $R(u)$ on $R_i$. Thus, we compute derivatives w.r.t. the $\mathbf{d}_j$

and then apply the chain rule. We focus on $\mathcal{L} = SO(3)$ as it is the most relevant group for computer vision applications.

In order to apply the chain rule, we need the derivatives of the $\mathbf{d}_j$ w.r.t. the $R_{i+j}$, which follow trivially from the definition of the right Jacobian and the adjoint of $SO(3)$:

$$\frac{\partial \mathbf{d}_j}{\partial R_{i+j}} = J_\mathrm{r}^{-1}(\mathbf{d}_j) R_{i+j}^\top \,, \quad \frac{\partial \mathbf{d}_{j+1}}{\partial R_{i+j}} = -\frac{\partial \mathbf{d}_{j+1}}{\partial R_{i+j+1}} \,. \tag{47}$$

Now consider a curve $\mathbf{f}$ that maps to $\mathbb{R}^d$, for example the spline value, velocity or acceleration. $\mathbf{f}$ depends on the set of control points $R_{i+j}$ and has derivatives

$$\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}R_{i+j}} = \frac{\partial \mathbf{f}}{\partial \mathbf{d}_j} \cdot \frac{\partial \mathbf{d}_j}{\partial R_{i+j}} + \frac{\partial \mathbf{f}}{\partial \mathbf{d}_{j+1}} \cdot \frac{\partial \mathbf{d}_{j+1}}{\partial R_{i+j}} \,. \tag{48}$$

for $j > 0$. For $j = 0$ we obtain

$$\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}R_i} = \frac{\partial \mathbf{f}}{\partial R_i} + \frac{\partial \mathbf{f}}{\partial \mathbf{d}_1} \cdot \frac{\partial \mathbf{d}_1}{\partial R_i} \,. \tag{49}$$

Thus, to compute Jacobians w.r.t. control points, we need the partial derivatives w.r.t. the $\mathbf{d}_j$ as well as $R_i$.

In the following, we will first derive some useful properties, and then present a recursive scheme to compute Jacobians of $\boldsymbol{\rho}$, $\boldsymbol{\omega} = \boldsymbol{\omega}^{(k)}$ and $\dot{\boldsymbol{\omega}} = \dot{\boldsymbol{\omega}}^{(k)}$, where we define the vector $\boldsymbol{\rho} \in \mathbb{R}^d$ as the mapping of $R$ to $\mathbb{R}^d$ by the Log map:

$$\boldsymbol{\rho}(u) = \operatorname{Log} R(u) \,. \tag{50}$$

The Jacobians of $\boldsymbol{\omega}^{(j)}$ and $\dot{\boldsymbol{\omega}}^{(j)}$ w.r.t. $\mathbf{d}_j$ are zero: from the recursion schemes of $\boldsymbol{\omega}$ and $\dot{\boldsymbol{\omega}}$ in Theorems 5.2 and 5.3, we find that the first index for which $\mathbf{d}_j$ appears explicitly or implicitly (in the form of $A_j$) is $j+1$.

Furthermore, we use the following important relation in our derivations, which is proven in the Appendix:

$$\frac{\partial}{\partial \mathbf{d}} \operatorname{Exp}(-\lambda \mathbf{d}) \boldsymbol{\omega} = \lambda \operatorname{Exp}(-\lambda \mathbf{d}) \boldsymbol{\omega}_\wedge J_\mathrm{r}(-\lambda \mathbf{d}) \tag{51}$$

for $\lambda \in \mathbb{R}$ and $\mathbf{d}, \boldsymbol{\omega} \in \mathbb{R}^3$. Together, these findings have two important implications:

**Theorem 5.4.** *The Jacobian of $\boldsymbol{\omega}^{(j+1)}$ w.r.t. $\mathbf{d}_j$ is*

$$\frac{\partial \boldsymbol{\omega}^{(j+1)}}{\partial \mathbf{d}_j} = \lambda_j A_j^\top \boldsymbol{\omega}_\wedge^{(j)} J_r(-\lambda_j \mathbf{d}_j) + \dot{\lambda}_j \mathbb{1} \,. \tag{52}$$

*Proof.* We apply (51) to $\boldsymbol{\omega}^{(j+1)}$ as defined in (38). □

**Theorem 5.5.** *The Jacobian of $\dot{\boldsymbol{\omega}}^{(j+1)}$ w.r.t. $\mathbf{d}_j$ is*

$$\begin{aligned} \frac{\partial \dot{\boldsymbol{\omega}}^{(j+1)}}{\partial \mathbf{d}_j} = \dot{\lambda}_j \left( \boldsymbol{\omega}_\wedge^{(j+1)} - D_j \frac{\partial \boldsymbol{\omega}^{(j+1)}}{\partial \mathbf{d}_j} \right) \\ + \lambda_j A_j^\top \dot{\boldsymbol{\omega}}_\wedge^{(j)} J_r(-\lambda_j \mathbf{d}_j) + \ddot{\lambda}_j \mathbb{1} \,. \end{aligned} \tag{53}$$

*Proof.* We apply (51) to $\dot{\boldsymbol{\omega}}^{(j+1)}$ as defined in (46) and use

$$\boldsymbol{\omega} \times \mathbf{d} = \boldsymbol{\omega}_\wedge \mathbf{d} = -D\boldsymbol{\omega} \tag{54}$$

for $\mathbf{d}, \boldsymbol{\omega} \in \mathbb{R}^3$ and $D = \mathbf{d}_\wedge$. □

These results for the Jacobians of $\boldsymbol{\omega}^{(j+1)}$ and $\dot{\boldsymbol{\omega}}^{(j+1)}$ w.r.t. $\mathbf{d}_j$ can be used to derive Jacobians of $\boldsymbol{\omega}$ and $\dot{\boldsymbol{\omega}}$ by recursion:

**Theorem 5.6.** *The following recurrence relation (from $j = k-1$ to $j = 1$) allows for Jacobian computation of $\boldsymbol{\rho}$, $\boldsymbol{\omega}$ and $\dot{\boldsymbol{\omega}}$ in a linear (w.r.t. $k$) number of multiplications and additions:*

$$P_{k-1} = \mathbb{1} \,, \tag{55}$$

$$\mathbf{s}_{k-1} = \mathbf{0} \,, \tag{56}$$

$$\frac{\partial \boldsymbol{\rho}}{\partial \mathbf{d}_j} = \lambda_j J_r^{-1}(\boldsymbol{\rho}) P_j J_r(\lambda_j \mathbf{d}_j) \,, \tag{57}$$

$$\frac{\partial \boldsymbol{\omega}}{\partial \mathbf{d}_j} = P_j \frac{\partial \boldsymbol{\omega}^{(j+1)}}{\partial \mathbf{d}_j} \,, \tag{58}$$

$$\frac{\partial \dot{\boldsymbol{\omega}}}{\partial \mathbf{d}_j} = P_j \frac{\partial \dot{\boldsymbol{\omega}}^{(j+1)}}{\partial \mathbf{d}_j} - (\mathbf{s}_j)_\wedge \frac{\partial \boldsymbol{\omega}}{\partial \mathbf{d}_j} \,, \tag{59}$$

$$P_{j-1} = P_j A_j^\top \,, \tag{60}$$

$$\mathbf{s}_{j-1} = \mathbf{s}_j + \dot{\lambda}_j P_j \mathbf{d}_j \,. \tag{61}$$

*$P_j$ and $\mathbf{s}_j$ are accumulator products and sums, respectively.*

*Proof of* (57). We write $R$ as

$$R = R_i A_\mathrm{pre} \operatorname{Exp}(\lambda_j \mathbf{d}_j) A_\mathrm{post} \,, \tag{62}$$

where $A_\mathrm{pre}$ and $A_\mathrm{post}$ are implicitly defined by comparison with (23) and do not depend on $\mathbf{d}_j$. The right Jacobian property (10), combined with the adjoint property, yields

$$\begin{aligned} &R_i A_\mathrm{pre} \operatorname{Exp}(\lambda_j (\mathbf{d}_j + \boldsymbol{\delta})) A_\mathrm{post} \\ &= R \operatorname{Exp}(\lambda_j A_\mathrm{post}^\top J_\mathrm{r}(\lambda_j \mathbf{d}_j) \boldsymbol{\delta}) + \mathcal{O}(\boldsymbol{\delta}^2) \end{aligned} \tag{63}$$

Now, we apply the right Jacobian property (9) to obtain

$$\begin{aligned} &\operatorname{Log}\left( R_i A_\mathrm{pre} \operatorname{Exp}(\lambda_j (\mathbf{d}_j + \boldsymbol{\delta})) A_\mathrm{post} \right) \\ &= \boldsymbol{\rho} + \lambda_j J_\mathrm{r}^{-1}(\boldsymbol{\rho}) A_\mathrm{post}^\top J_\mathrm{r}(\lambda_j \mathbf{d}_j) \boldsymbol{\delta} + \mathcal{O}(\boldsymbol{\delta}^2) \,. \end{aligned} \tag{64}$$

Differentiation at $\boldsymbol{\delta} = 0$ and inserting $A_\mathrm{post}^\top = P_j$ yields (57). □

*Proof of* (58). Since $j \leq k-2$, $A_{k-1}$ does not depend on $\mathbf{d}_j$, thus

$$\begin{aligned} \frac{\partial \boldsymbol{\omega}}{\partial \mathbf{d}_j} &= \frac{\partial}{\partial \mathbf{d}_j} \left( A_{k-1}^\top \boldsymbol{\omega}^{(k-1)} + \dot{\lambda}_{k-1} \mathbf{d}_{k-1} \right) \\ &= A_{k-1}^\top \frac{\partial \boldsymbol{\omega}^{(k-1)}}{\partial \mathbf{d}_j} \,. \end{aligned} \tag{65}$$

Iterative application of this equation leads to the claim. □

*Proof of* (59). First, since the case $j = k-1$ is trivial, we can focus on $j \leq k-2$: for these cases, we find (by insertion into (46) that

$$\begin{aligned}
&\frac{\partial \dot{\boldsymbol{\omega}}}{\partial \mathbf{d}_j} \\
&= \frac{\partial}{\partial \mathbf{d}_j} \left( -\dot{\lambda}_{k-1} D_{k-1} \boldsymbol{\omega} + A_{k-1}^\top \dot{\boldsymbol{\omega}}^{(k-1)} + \ddot{\lambda}_{k-1} \mathbf{d}_{k-1} \right) \\
&= -\dot{\lambda}_{k-1} D_{k-1} \frac{\partial \boldsymbol{\omega}}{\partial \mathbf{d}_j} + A_{k-1}^\top \frac{\partial \dot{\boldsymbol{\omega}}^{(k-1)}}{\partial \mathbf{d}_j} \,.
\end{aligned} \tag{66}$$

We prove the equivalence of this and (59) by induction in the Appendix. □

## 6. Experiments

To evaluate the proposed formulation for the B-spline time derivatives and our $SO(3)$ Jacobians, we conduct two experiments. In the first one simulated velocity and acceleration measurements are used to estimate the trajectory represented by the spline. This allows us to highlight the computational advantages of the proposed formulation. In the second experiment we demonstrate an example of a real-world application, in particular a multiple camera and IMU calibration. In this case we estimate the continuous trajectory of the IMU, transformations from the cameras to the IMU, accelerometer and gyroscope biases and gravity in the world frame.

In both cases, the baseline method for comparison computes time derivatives as used in prior work [13, 14]. Unless stated otherwise, optimizations are done using Ceres [1] with the automatic differentiation option. This option uses dual numbers for computing Jacobians. In all cases we use the Levenberg-Marquardt algorithm for optimization with sparse Cholesky decomposition for solving linear systems. The experiments were conducted on Ubuntu 18.04 with Intel Xeon E5-1620 CPU. We used clang-9 as a compiler with `-O3 -march=native -DNDEBUG` flags. Even though residual and Jacobian computations are easily parallelizable, in this paper we concentrate on differences between formulations and run all experiments in single-thread configuration.

We have made the experiments available open-source at: https://gitlab.com/tum-vision/lie-spline-experiments

### 6.1. Simulated Sequence

One typical application of B-splines on Lie groups is trajectory estimation from a set of sensor measurements. In our first experiment we assume that we have pose, velocity and acceleration measurements for either $SO(3)$ or $SE(3)$ and formulate an optimization problem that is supposed to estimate the values of the spline knots representing the true trajectory. In this case we minimize the sum of squared residuals, where a residual is the difference between the measured and the computed value.

| $\mathcal{L}$ | $k$ | Config. | Ours | Baseline | Speedup |
|---|---|---|---|---|---|
| $SE(3)$ | 5 | acc. | **0.445** | 1.196 | 2.69 |
| $SE(3)$ | 5 | vel. | **0.405** | 0.581 | 1.43 |
| $SE(3)$ | 6 | acc. | **0.644** | 2.332 | 3.62 |
| $SE(3)$ | 6 | vel. | **0.590** | 0.936 | 1.59 |
| $SO(3)$ | 5 | acc. | **0.081** | 0.280 | 3.45 |
| $SO(3)$ | 5 | vel. | **0.082** | 0.141 | 1.73 |
| $SO(3)$ | 6 | acc. | **0.117** | 0.520 | 4.43 |
| $SO(3)$ | 6 | vel. | **0.111** | 0.217 | 1.95 |

Table 1. Optimization time in seconds for the proposed and baseline formulations with velocity (*vel.*) and acceleration (*acc.*) measurements, and the speedup achieved by our formulation. In all the experiments both formulations converged to the same result with the same number of iterations.

We use a spline with $100 + k$ knots with 2 second spacing, 25 value measurements and 2020 velocity or acceleration measurements that are uniformly distributed across the spline. The measurements are generated from the ground-truth spline. We initialize the knot values of the splines that will be optimized to perturbed ground truth values, which results in 5 optimization iterations until convergence. Table 1 summarizes the results. As expected, the proposed formulation outperforms the baseline formulation in all cases. The time difference is higher for the acceleration measurements, since there the baseline formulation is cubic in the order of spline.

### 6.2. Camera-IMU calibration

In the second experiment we aim to show the applicability of our approach for real-world applications with camera-IMU calibration as an example. We use two types of splines of order 5 and knot spacing of 10 ms to represent the continuous motion of the IMU coordinate frame: $SO(3) \times \mathbb{R}^3$ (split representation) and $SE(3)$. For both cases we implemented the proposed and the baseline method to compute time derivatives.

We use the *calib-cam1* sequence of [19] in this experiment. It contains 51.8 seconds of data and consists of 10336 accelerometer and the same number of gyroscope measurements, 1036 images for two cameras which observe 291324 corners of the calibration pattern. We assume that the camera intrinsic parameters are pre-calibrated and there is a good initial guess between camera and IMU rotations computed from rotational velocities. All optimizations in our experiment have the same initial conditions and noise settings. A segment of the sequence trajectory after optimization is visualized in Figure 1.

| Estimated variable | $\mathbf{g}$ [m/s$^2$] | $\mathbf{b}_a$ [m/s$^2$] | $\mathbf{b}_g$ [rad/s] | $\mathbf{t}_{ic0}$ [m] | $\mathbf{t}_{ic1}$ [m] | $R_{ic0}$ [rad] | $R_{ic1}$ [rad] |
|---|---|---|---|---|---|---|---|
| Max deviation | $6.07 \cdot 10^{-5}$ | $6.32 \cdot 10^{-5}$ | $2.14 \cdot 10^{-9}$ | $6.34 \cdot 10^{-6}$ | $6.33 \cdot 10^{-6}$ | $3.51 \cdot 10^{-8}$ | $3.34 \cdot 10^{-8}$ |

Table 2. Maximum difference between the mean estimate and the estimates from all calibration methods. For vectors ($\mathbf{g}$, $\mathbf{b}_a$, $\mathbf{b}_g$, $\mathbf{t}_{ic0}$, $\mathbf{t}_{ic1}$), the $L_2$ norm is used. For rotational values ($R_{ic0}$, $R_{ic1}$) the angle norm in radians is used. The results show that independent of the underlying spline representation ($SO(3) \times \mathbb{R}^3$ or $SE(3)$) the calibration converges to the same result.

The projection residuals are defined as:

$$\mathbf{r}_p(u) = \pi(T_{ic}^{-1} T_{wi}(u)^{-1} \mathbf{x}) - \hat{\mathbf{p}}, \tag{67}$$

$$T_{wi} = \begin{pmatrix} R_{wi} & \mathbf{t}_{wi} \\ 0 & 1 \end{pmatrix} \in SE(3), \tag{68}$$

where $T_{wi}(u)$ is the pose of the IMU in the coordinate frame computed from the spline either as a pose directly ($SE(3)$), or as two separate values for rotation and translation ($SO(3) \times \mathbb{R}^3$). $T_{ic}$ is the transformation from the camera where the corner was observed to the IMU, $\pi$ is a fixed projection function, $\mathbf{x}$ is the 3D coordinate of the calibration corner and $\hat{\mathbf{p}}$ denotes the 2D coordinates of the corner detected in the image.

The gyroscope and accelerometer residuals are defined as:

$$\mathbf{r}_\omega(u) = \boldsymbol{\omega}(u) - \tilde{\boldsymbol{\omega}} + \mathbf{b}_g, \tag{69}$$

$$\mathbf{r}_a(u) = R_{wi}(u)^{-1}(\ddot{\mathbf{t}}_{wi}(u) + \mathbf{g}) - \tilde{\mathbf{a}} + \mathbf{b}_a, \tag{70}$$

where $\tilde{\boldsymbol{\omega}}$ and $\tilde{\mathbf{a}}$ are the measurements, $\mathbf{b}_g$ and $\mathbf{b}_a$ are static biases and $\mathbf{g}$ is the gravity vector in world coordinates. $R_{wi}(u)$ is the rotation from IMU to world frame. The definition of $\boldsymbol{\omega}(u)$ and $\ddot{\mathbf{t}}_{wi}(u)$ depends on the spline representation that we use. For the $SO(3) \times \mathbb{R}^3$ representation, $\boldsymbol{\omega}(u)$ is the rotational velocity in the body frame computed as in (38) and $\ddot{\mathbf{t}}_{wi}(u)$ is the second derivative of the $\mathbb{R}^3$ spline representing the translation of the IMU in the world frame. For $SE(3)$, $\boldsymbol{\omega}(u)$ is the rotational component of the velocity computed in (34) and $\ddot{\mathbf{t}}_{wi}(u)$ is the translation part of the second time derivative of the pose computed in (40). The $SE(3)$ formulation of these residuals is identical to the one used in [13].

The calibration is done by minimizing a function that combines the residuals for all measurements:

$$E = \sum \mathbf{r}_\omega^\top W_\omega \mathbf{r}_\omega + \sum \mathbf{r}_a^\top W_a \mathbf{r}_a + \sum \mathbf{r}_p^\top W_p \mathbf{r}_p, \tag{71}$$

where $W_\omega, W_a, W_p$ are the weight matrices computed using the sensor noise characteristics.

In all conducted experiments the calibration converged to the same values (see Table 2) after 12 iterations. Our results confirm previous reports [9, 16] that the $SE(3)$ spline representation does not introduce any advantages compared to the split representation, but requires more computations.

The timing results are presented in Table 3. In all of the cases we can see the advantage of the proposed formulation for time derivatives. In the case of split representation only the gyroscope residuals are affected, so the difference is relatively small if Ceres Jacobians are used (6% less time). For the $SE(3)$ representation, both gyroscope and accelerometer residuals are affected, since we need to compute the second time derivative for linear acceleration. In this case our formulation results in 36.7% less computation time. We also present the results with our custom solver that uses split representation and the analytic Jacobians for $SO(3)$ that we introduced in Section 5.2. It results in a further decrease in the computation time and is able to perform the calibration in less than 6 seconds (2.6 times faster than the baseline approach with split representation).

| $SO(3) \times \mathbb{R}^3$ | | | $SE(3)$ | |
|---|---|---|---|---|
| Ours Analytic | Ours Ceres | Baseline Ceres | Ours Ceres | Baseline Ceres |
| 5.82 | 14.18 | 15.09 | 23.56 | 37.14 |

Table 3. Time in seconds to perform the camera-IMU calibration. Analytic uses a custom solver with the analytic Jacobians for all residuals. All other methods use Ceres solver with dual numbers for Jacobian computations.

The results indicate that our formulation of the time derivatives requires less computations, especially if second time derivatives need to be computed. This can have an even larger effect for the calibration of multiple IMUs [18], where even for the split formulation, evaluation of the rotational acceleration is required.

## 7. Conclusions

In this work, we showed how commonly used B-splines on Lie groups can be differentiated (w.r.t. time and control points) in a very efficient way. Both our temporal derivatives and our Jacobians can be computed in $\mathcal{O}(k)$ matrix operations, while traditional computation schemes are up to cubic in $k$. We mathematically prove the correctness of our statements. While our contribution has a clear focus on theory, we also demonstrate how our derivatives lead to speedups of up to 4.4x in practical computer vision applications. This makes our proposed method highly relevant for real-time applications of continuous-time trajectory representations.

This work was supported by the ERC Consolidator Grant "3D Reloaded".

## References


[1] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. http://ceres-solver.org. 7

[2] Timothy D Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017. 2

[3] Gregory S Chirikjian. *Stochastic Models, Information Theory, and Lie Groups, Volume 2: Analytic Methods and Modern Applications*, volume 2. Springer Science & Business Media, 2011. 3, 10

[4] Maurice G Cox. The numerical evaluation of B-splines. *IMA Journal of Applied Mathematics*, 1972. 2, 3

[5] Carl De Boor. On calculating with B-splines. *Journal of Approximation theory*, 1972. 2, 3

[6] Paul de Casteljau. Courbes à pôles, 1959. 2

[7] W. Ding, W. Gao, K. Wang, and S. Shen. An efficient B-spline-based kinodynamic replanning framework for quadrotors. *IEEE Transactions on Robotics*, pages 1–20, 2019. 2

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

[9] Adrian Haarbach, Tolga Birdal, and Slobodan Ilic. Survey of higher order rigid body motion interpolation methods for keyframe animation and continuous-time trajectory estimation. In *2018 International Conference on 3D Vision (3DV)*, pages 381–389. IEEE, 2018. 2, 8

[10] Alan Jeffrey and Daniel Zwillinger. *Table of integrals, series, and products*. Elsevier, 2007. 10

[11] C. Kerl, J. Stückler, and D. Cremers. Dense continuous-time tracking and mapping with rolling shutter RGB-D cameras. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2264–2272, Dec 2015. 2, 4

[12] Myoung-Jun Kim, Myung-Soo Kim, and Sung Yong 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*, SIGGRAPH '95, pages 369–376, New York, NY, USA, 1995. ACM. 2, 4

[13] Steven Lovegrove, Alonso Patron-Perez, and Gabe Sibley. Spline Fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras. In *Proc. British Mach. Vis. Conf.*, page 93.1–93.12, 2013. 2, 4, 5, 7, 8

[14] E. Mueggler, G. Gallego, H. Rebecq, and D. Scaramuzza. Continuous-time visual-inertial odometry for event cameras. *IEEE Transactions on Robotics*, 34(6):1425–1440, Dec 2018. 2, 4, 5, 7

[15] Hannes Ovrén and Per-Erik Forssén. Spline error weighting for robust visual-inertial fusion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 321–329, 2018. 2, 4

[16] Hannes Ovrén and Per-Erik Forssén. Trajectory representation and landmark projection for continuous-time structure from motion. *The International Journal of Robotics Research*, 38(6):686–701, 2019. 2, 4, 5, 8

[17] Kaihuai Qin. General matrix representations for B-splines. *The Visual Computer*, 16(3-4):177–186, 2000. 2, 3

[18] J. Rehder, J. Nikolic, T. Schneider, T. Hinzmann, and R. Siegwart. Extending kalibr: Calibrating the extrinsics of multiple IMUs and of individual axes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4304–4311, May 2016. 8

[19] D. Schubert, T. Goll, N. Demmel, V. Usenko, J. Stückler, and D. Cremers. The TUM VI benchmark for evaluating visual-inertial odometry. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1680–1687, Oct 2018. 7

[20] Joan Solà, Jeremie Deray, and Dinesh Atchuthan. A micro Lie theory for state estimation in robotics. *arXiv preprint arXiv:1812.01537*, 2018. 2

[21] Hannes Sommer, James Richard Forbes, Roland Siegwart, and Paul Furgale. Continuous-time estimation of attitude using B-splines on Lie groups. *Journal of Guidance, Control, and Dynamics*, 39(2):242–261, 2015. 2, 4

[22] Vladyslav Usenko, Lukas von Stumberg, Andrej Pangercic, and Daniel Cremers. Real-time trajectory replanning for MAVs using uniform B-splines and a 3D circular buffer. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017. 2

# Supplemental Material: Efficient Derivative Computation for Cumulative B-Splines on Lie Groups

## 8. Appendix

### 8.1. Right Jacobian for $SO(3)$

If $\mathcal{L} = SO(3)$, the right Jacobian and its inverse can be found in [3, p. 40]:

$$J_\mathrm{r}(\mathbf{x}) = \mathbb{1} - \frac{1-\cos(\|\mathbf{x}\|)}{\|\mathbf{x}\|^2}\mathbf{x}_\wedge + \frac{\|\mathbf{x}\| - \sin(\|\mathbf{x}\|)}{\|\mathbf{x}\|^3}\mathbf{x}_\wedge^2\,, \tag{72}$$

$$J_\mathrm{r}(\mathbf{x})^{-1} = \mathbb{1} + \frac{1}{2}\mathbf{x}_\wedge + \left(\frac{1}{\|\mathbf{x}\|^2} - \frac{1+\cos(\|\mathbf{x}\|)}{2\|\mathbf{x}\|\sin(\|\mathbf{x}\|)}\right)\mathbf{x}_\wedge^2\,. \tag{73}$$

### 8.2. Third order time derivatives

For completeness, we state the third order time derivatives for a general Lie group $\mathcal{L}$ here. The proofs are in analogy to those of the first and second order time derivatives, thus we do not repeat them here.

$$\dddot{R} = R\left((\boldsymbol{\omega}_\wedge^{(k)})^3 + 2\boldsymbol{\omega}_\wedge^{(k)}\dot{\boldsymbol{\omega}}_\wedge^{(k)} + \dot{\boldsymbol{\omega}}_\wedge^{(k)}\boldsymbol{\omega}_\wedge^{(k)} + \ddot{\boldsymbol{\omega}}_\wedge^{(k)}\right)\,, \tag{74}$$

$$\ddot{\boldsymbol{\omega}}^{(j)} = \mathrm{Adj}_{A_{j-1}^{-1}}\,\ddot{\boldsymbol{\omega}}^{(j-1)} + \dddot{\lambda}_{j-1}\mathbf{d}_{j-1} + \left[\ddot{\lambda}_{j-1}\boldsymbol{\omega}_\wedge^{(j)} + 2\dot{\lambda}_{j-1}\dot{\boldsymbol{\omega}}_\wedge^{(j)} - \dot{\lambda}_{j-1}^2[\boldsymbol{\omega}_\wedge^{(j)}, D_{j-1}], D_{j-1}\right]_\vee\,, \tag{75}$$

$$\ddot{\boldsymbol{\omega}}^{(1)} = \mathbf{0} \in \mathbb{R}^d\,. \tag{76}$$

$\ddot{\boldsymbol{\omega}}$ is called *jerk*. For $\mathcal{L} = SO(3)$, the expression (75) becomes slightly simpler:

$$\ddot{\boldsymbol{\omega}}^{(j)} = A_{j-1}^\top\ddot{\boldsymbol{\omega}}^{(j-1)} + \dddot{\lambda}_{j-1}\mathbf{d}_{j-1} + \left(\ddot{\lambda}_{j-1}\boldsymbol{\omega}^{(j)} + 2\dot{\lambda}_{j-1}\dot{\boldsymbol{\omega}}^{(j)} - \dot{\lambda}_{j-1}^2\boldsymbol{\omega}^{(j)}\times\mathbf{d}_{j-1}\right)\times\mathbf{d}_{j-1}\,. \tag{77}$$

### 8.3. Complexity analysis

While we are not the first to write down temporal derivatives of Lie group splines, we actually are the first to compute these in only $\mathcal{O}(k)$ matrix operations (multiplications and additions). Additionally, to the best of our knowledge, we are the first to explicitly propose a scheme for Jacobian computation in $SO(3)$, which also does not need more than $\mathcal{O}(k)$ matrix operations. In Table 4, we provide an overview of the needed number of multiplications and additions for the temporal derivatives (both in related work and according to the proposed formulation).

### 8.4. Proofs

#### 8.4.1 Proof of (20) (cumulative blending matrix)

After substituting summation indices $s \leftarrow k-1-s$ and $l \leftarrow l-s$ we get

$$\widetilde{m}_{0,n}^{(k)} = \frac{C_{k-1}^n}{(k-1)!}\sum_{s=0}^{k-1}\sum_{l=0}^{s}(-1)^l C_k^l (s-l)^{k-1-n}\,. \tag{78}$$

We now show by induction over $k$ that $\widetilde{m}_{0,n}^{(k)} = \delta_{n,0}$ for all $n = 0, ..., k-1$: for $k=1$, $\widetilde{m}_{0,n}^{(k)} = 1 = \delta_{0,0}$ is trivial. Now, assume $\widetilde{m}_{0,n}^{(k)} = \delta_{n,0}$ for some $k$.

Starting from the induction assumption

$$\sum_{s=0}^{k-1}\sum_{l=0}^{s}(-1)^l C_k^l (s-l)^{k-1-n} = (k-1)!\,\delta_{n,0} \tag{79}$$

we now show that $\widetilde{m}_{0,n}^{(k+1)} = \delta_{n,0}$ for $n = 0, ..., k-1$. If not indicated otherwise, we use well-known binomial sum properties, as summarized in e.g. [10, 0.15]. As a first step, we use the property $C_{k+1}^l = C_k^l + C_k^{l-1}$ and split the terms in the double sum to obtain

$$\sum_{s=0}^{k}\sum_{l=0}^{s}(-1)^l C_{k+1}^l (s-l)^{k-n} = T_1 + T_2 + T_3 + T_4\,, \tag{80}$$

with

$$T_1 = \sum_{s=0}^{k-1}\sum_{l=0}^{s}(-1)^l C_k^l (s-l)^{k-n}\,, \tag{81}$$

$$T_2 = \sum_{l=0}^{k}(-1)^l C_k^l (k-l)^{k-n}\,, \tag{82}$$

$$T_3 = \sum_{s=0}^{k}\sum_{l=0}^{s-1}(-1)^l C_k^{l-1} (s-l)^{k-n}\,, \tag{83}$$

$$T_4 = \sum_{s=0}^{k}(-1)^s C_k^{s-1}(s-s)^{k-n}\,. \tag{84}$$

It is easy to see from (79) that $T_1 = (k-1)!\,\delta_{n,1}$. Furthermore, $T_2 = k!\,\delta_{n,0}$. For $T_4$, we have

$$\begin{aligned} T_4 &= 0^{k-n}\sum_{s=0}^{k}(-1)^s C_k^{s-1} = \delta_{n,k}\sum_{s=0}^{k-1}(-1)^{s+1}C_k^s \\ &= -\delta_{n,k}(-1)^{k-1}C_{k-1}^{k-1} = \delta_{n,k}(-1)^k\,. \end{aligned} \tag{85}$$

| | $\dot{R}$ Baseline | | $\dot{R}$ Ours | | $\ddot{R}$ Baseline | | $\ddot{R}$ Ours, any $\mathcal{L}$ | | $\ddot{R}$ Ours, $SO(3)$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| m-m mult. | $(k-1)^2+1$ | 10 | 1 | 1 | $\frac{1}{2}k^2(k-1)$ | 24 | $2k$ | 8 | 2 | 2 |
| m-v mult. | 0 | 0 | $k-1$ | 3 | 0 | 0 | $k-1$ | 3 | $2(k-1)$ | 6 |
| add. | $k-2$ | 2 | $k-1$ | 3 | $\frac{1}{2}k^2(k-1)$ | 24 | $3k-2$ | 10 | $2k-1$ | 7 |

Table 4. Number of matrix operations needed to compute temporal derivatives of the $\mathcal{L}$-valued splines: *m-m/m-v mult.* denote matrix-matrix and matrix-vector multiplications, respectively. *add.* denotes additions of matrices or vectors. Our formulation needs consistently less operations than the baseline approach. The blue numbers give the number of operations for a cubic spline ($k=4$).

Finally, we need $T_3$, which is the most complicated term:

$$\begin{aligned} T_3 &= \sum_{s=0}^{k}\sum_{l=0}^{s-1}(-1)^l C_k^{l-1}(s-l)^{k-n} \\ &= \sum_{s=0}^{k-1}\sum_{l=0}^{s-1}(-1)^{l+1} C_k^{l}(s-l)^{k-n} \\ &= -\sum_{s=0}^{k-1}\sum_{l=0}^{s}(-1)^l C_k^{l}(s-l)^{k-n} + \sum_{s=0}^{k-1} C_k^s 0^{k-n} \\ &= -(k-1)!\,\delta_{n,1} + \delta_{n,k}(-1)^{k-1}\,, \end{aligned} \tag{86}$$

where the first equality comes from index shifting ($s \leftarrow s-1$ and $l \leftarrow l-1$), and the last one uses the induction assumption. In total, we obtain:

$$\begin{aligned} &T_1 + T_2 + T_3 + T_4 \\ &= (k-1)!\,\delta_{n,1} + k!\,\delta_{n,0} \\ &\quad - (k-1)!\,\delta_{n,1} - \delta_{n,k}(-1)^k + \delta_{n,k}(-1)^k \\ &= k!\,\delta_{n,0}\,, \end{aligned} \tag{87}$$

which is equivalent to $\widetilde{m}_{0,n}^{(k+1)} = \delta_{n,0}$ for $n = 0, ..., k-1$ by definition of $\widetilde{m}_{j,n}^{(k+1)}$.

What remains is the case $n=k$:

$$\sum_{s=0}^{k}\sum_{l=0}^{s}(-1)^l C_{k+1}^{l}(s-l)^0 = \sum_{s=0}^{k}(-1)^s C_k^s = 0\,, \tag{88}$$

which concludes the proof.

#### 8.4.2 Proof of (51) (Jacobian of $\mathrm{Exp}(-\lambda\mathbf{d})$ multiplied by a vector)

$$\begin{aligned} &\frac{\partial}{\partial \mathbf{d}}\mathrm{Exp}(-\lambda\mathbf{d})\boldsymbol{\omega} = \frac{\partial}{\partial \boldsymbol{\delta}}\mathrm{Exp}(-\lambda(\mathbf{d}+\boldsymbol{\delta}))\boldsymbol{\omega}\Big|_{\boldsymbol{\delta}=0} \\ &= \frac{\partial}{\partial \boldsymbol{\delta}}\left(\mathrm{Exp}(-\lambda\mathbf{d})\,\mathrm{Exp}(-J_\mathrm{r}(-\lambda\mathbf{d})\lambda\boldsymbol{\delta})\boldsymbol{\omega} + \mathcal{O}(\boldsymbol{\delta}^2)\right)\Big|_{\boldsymbol{\delta}=0} \\ &= -\lambda\,\mathrm{Exp}(-\lambda\mathbf{d})\,\frac{\partial}{\partial\boldsymbol{\delta}}(\mathrm{Exp}(\boldsymbol{\delta})\boldsymbol{\omega})\Big|_{\boldsymbol{\delta}=0} J_\mathrm{r}(-\lambda\mathbf{d}) \\ &= \lambda\,\mathrm{Exp}(-\lambda\mathbf{d})\boldsymbol{\omega}_\wedge J_\mathrm{r}(-\lambda\mathbf{d})\,. \end{aligned} \tag{89}$$

For the second equality we have used the right Jacobian property (10). To obtain the last equality, note that

$$\frac{\partial\,\mathrm{Exp}(\boldsymbol{\delta})\boldsymbol{\omega}}{\partial\boldsymbol{\delta}}\Big|_{\boldsymbol{\delta}=0} = -\boldsymbol{\omega}_\wedge\,. \tag{90}$$

#### 8.4.3 Proof of (59) (Jacobian of acceleration)

We show by induction that the following two formulas are equivalent for $l = j+2, \ldots, k$:

$$\frac{\partial\dot{\boldsymbol{\omega}}^{(l)}}{\partial\mathbf{d}_j} = -\dot{\lambda}_{l-1}D_{l-1}\frac{\partial\boldsymbol{\omega}^{(l)}}{\partial\mathbf{d}_j} + A_{l-1}^\top\frac{\partial\dot{\boldsymbol{\omega}}^{(l-1)}}{\partial\mathbf{d}_j}\,, \tag{91}$$

$$\frac{\partial\dot{\boldsymbol{\omega}}^{(l)}}{\partial\mathbf{d}_j} = P_j^{(l)}\frac{\partial\dot{\boldsymbol{\omega}}^{(j+1)}}{\partial\mathbf{d}_j} - (\mathbf{s}_j^{(l)})_\wedge\frac{\partial\boldsymbol{\omega}^{(l)}}{\partial\mathbf{d}_j}\,, \tag{92}$$

where we define $P_j^{(l)}$ and $\mathbf{s}_j^{(l)}$ as

$$P_j^{(l)} = \left(\prod_{m=j+1}^{l-1}A_m\right)^\top \quad\Rightarrow\quad P_j^{(k)} = P_j\,, \tag{93}$$

$$\mathbf{s}_j^{(l)} = \sum_{m=j+1}^{l-1}\dot{\lambda}_m P_m\mathbf{d}_m \quad\Rightarrow\quad \mathbf{s}_j^{(k)} = \mathbf{s}_j\,. \tag{94}$$

The case $l=k$ then is the desired results. For $l = j+2$, we easily see that

$$-\dot{\lambda}_{l-1}D_{l-1} = -(\dot{\lambda}_{j+1}\mathbf{d}_{j+1})_\wedge = -(\mathbf{s}_j^{(l)})_\wedge\,, \tag{95}$$

$$A_{l-1}^\top = A_{j+1}^\top = P_j^{(j+2)} = P_j^{(l)}\,, \tag{96}$$

which together implies the equivalence of (91) and (92). Now, assume the equivalence holds for some $l \in \{j+2, \ldots, k-1\}$ and note that

$$A_l^\top P_j^{(l)} = P_j^{(l+1)}\,, \tag{97}$$

$$A_l^\top(s_j^{(l)})_\wedge = \left((s_j^{(l+1)})_\wedge - \dot{\lambda}_l D_l\right)A_l^\top\,. \tag{98}$$

Then, starting from (91) and using the induction assumption, we obtain

$$\begin{aligned}\frac{\partial \dot{\boldsymbol{\omega}}^{(l+1)}}{\partial \mathbf{d}_j} &= -\dot{\lambda}_l D_l \frac{\partial \boldsymbol{\omega}^{(l+1)}}{\partial \mathbf{d}_j} \\ &\quad + A_l^\top \left( P_j^{(l)} \frac{\partial \dot{\boldsymbol{\omega}}^{(j+1)}}{\partial \mathbf{d}_j} - (\mathbf{s}_j^{(l)})_\wedge \frac{\partial \boldsymbol{\omega}^{(l)}}{\partial \mathbf{d}_j} \right) \\ &= -\dot{\lambda}_l D_l \frac{\partial \boldsymbol{\omega}^{(l+1)}}{\partial \mathbf{d}_j} + P_j^{(l+1)} \frac{\partial \dot{\boldsymbol{\omega}}^{(j+1)}}{\partial \mathbf{d}_j} \\ &\quad - \left( (s_j^{(l+1)})_\wedge - \dot{\lambda}_l D_l \right) A_l^\top \frac{\partial \boldsymbol{\omega}^{(l)}}{\partial \mathbf{d}_j} \,.\end{aligned} \tag{99}$$

The first and the last summand cancel, and what remains is (92) for $l+1$, which concludes the proof.