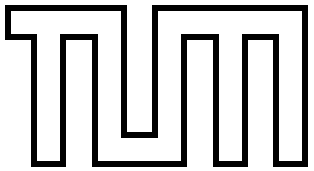


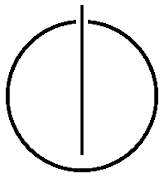
FAKULTÄT FÜR INFORMATIK

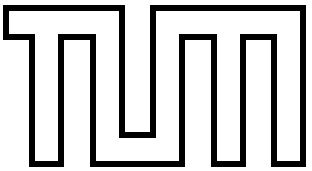
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

# Continuous time trajectory estimation for 3D SLAM from an actuated 2D laser scanner

Adrian Haarbach





FAKULTÄT FÜR INFORMATIK

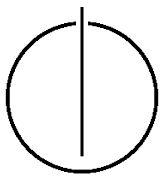
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

# Continuous time trajectory estimation for 3D SLAM from an actuated 2D laser scanner

# Zeitkontinuierliche Trajektorienschätzung für 3D-SLAM von einem sich bewegenden 2D-Laserscanner


| | |
|---|---|
| Author: | Adrian Haarbach |
| Supervisor: | Prof. Dr. Rüdiger Westermann |
| Advisor: | Dr. Stefan Romberg |
| Date: | December 15, 2016 |

I assure the single handed composition of this master's thesis only supported by declared resources.

Ich versichere, dass ich diese Master's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Munich, December 15, 2016 Adrian Haarbach

# Acknowledgments

First of all, I would like to thank Prof. Dr. Rüdiger Westermann for providing me with the opportunity to pursue my master's thesis at his computer graphics chair. I am also very grateful for my advisor Dr. Stefan Romberg who offered me the chance to do an exciting internship at NavVis. It was him who suggested the challenging and fascinating topic pursued in this thesis.

The internship gave me the chance to meet many great people who helped me in the struggle for thesis completion. A big thank you to my mentor Paul Zeller who gave me a head start with his awesome simulator and who never got tired of all my questioning when drafting the algorithm. I would also like to thank my colleagues Humberto Alvarez for his calibration knowledge and chocolate supply, Tim Habigt for his insights in mathematical optimization, Christian Werner for Americanizing my English and Octavio Cervantes for his hardware knowledge and cooking skills.

I am standing on the shoulders of giants and would thus like to thank specific representatives of the Open Source community as well as the academic world. The former for providing great software such as Ceres and Eigen upon which I could build on, represented by their maintainers Sameer Agarwal and Gaël Guennebaud who immediately answered all the questions I asked via mailing lists. The latter for being responsive to my emails concerning publication details like Erik B. Dam and Per-Erik Forssén were.

Finally, I happily repeat acknowledgments I already made in my bachelor's thesis, addressing the constants in my life: I would like to thank my family for the support during my studies, my father for proof-reading this thesis, my friends for reminding me of the fun part of life and my beloved Florian for his patience, care and support.

# Abstract

This thesis aims at estimating trajectories for 3D SLAM applications. A continuous time formulation allows solving multiple problems inherent to the traditional discrete time approaches seamlessly, such as sensor fusion of actuated 2D laser scanner data with inertial measurements. Special care is taken when choosing an appropriate trajectory representation. The well-known ICP algorithm used for rigid registration is extended significantly so it can deal with continuous-time, multi-view registration of deformable scans. The resulting algorithm can be employed online in a time-windowed fashion to get an open-loop trajectory estimate and offline for global optimization to further reduce the drift. In contrast to previous work we are able to provide ground truth data for evaluation by extending an existing simulator so that it can simulate actuated 2D laser scanner data with corresponding inertial measurements. Experiments on synthetic data with different scenarios, noise levels and parameter settings show the versatility, stability and adaptability of our algorithm as well as its high overall accuracy.

# Abbreviations

| | |
|---|---|
| **ATE** | Absolute Trajectory Error |
| **CICP** | Continuous ICP |
| **CT-SLAM** | Continuous time SLAM |
| **DoF** | Degree(s) of Freedom |
| **GICP** | Generalized ICP |
| **ICP** | Iterative Closest Point |
| **IMU** | Inertial Measurement Unit |
| **LERP** | **L**inear int**erp**olation |
| **LiDAR** | **Li**ght **D**etection **A**nd **R**anging |
| **LM** | Levenberg-Marquardt |
| **LM-ICP** | Levenberg-Marquardt ICP |
| **MV-LM-ICP** | Multiview Levenberg-Marquardt ICP |
| **NICP** | Normal ICP |
| **NLS** | Non-linear Least Squares |
| **PCA** | Principal Component Analysis |
| **RMS** | Root Mean Squared |
| **RPE** | Relative Pose Error |
| **SLAM** | Simultaneous Localization And Mapping |
| **SLERP** | **S**pherical **l**inear quaternion int**erp**olation |
| **SE(3)** | Special Euclidean group of rigid body motions |
| **se(3)** | Lie algebra of twists |
| **SO(3)** | Special orthogonal group of rotation matrices |
| **so(3)** | Lie algebra of skew-symmetric 3x3 matrices |
| **SQUAD** | **S**pherical cubic spline **quad**rangle quaternion interpolation |
| **SU(2)** | Special unitary group isomorphic to unit quaternions |
| **su(2)** | Lie algebra isomorphic to pure quaternions |
| **surfel** | **surf**ace **el**ement |
| **SVD** | Singular Value Decomposition |
| **sweep** | data collected in one half-revolution |
| **VO** | Visual Odometry |

# Contents

# 1 Introduction


(a) stationary 3D laser scanner Faro Focus3d-X130.[1]

(b) actuated 2D laser scanner SICK LMS-291 mounted on a vehicle. [BZ09]

Figure 1.1: Choices when digitizing the environment.

Digitization of indoor space is currently a hot topic in areas such as construction planning or interactive retail. The most accurate digital models of scenes were historically acquired with high accuracy 3D laser scanners (fig. 1.1a) from the surveying domain. However these stationary solutions have several disadvantages when acquiring large-scale models. The used sensor takes a lot of time to gather all the data of the surrounding environment. During the acquisition of a scan it must furthermore stay fixed to avoid deformations in the measured point clouds. However, to cover the whole environment, multiple measurements have to be taken at various locations inside the scene and subsequently be aligned to get a complete model of the scene. This stop-and go approach is very time-consuming and requires the final alignment of multiple point clouds by hand.

On the other hand, Simultaneous Localization And Mapping (SLAM) algorithms originating from the robotics domain allow to continuously estimate a trajectory from a moving platform while building a map of the environment. These have originally been limited to 2D, but recent progress in sensor hardware and algorithms together with the increased computational power available allow for full 3D SLAM achieving nearly the accuracy of the stationary solutions, but in far less time.

While there are specifically developed 3D laser scanners for 3D SLAM applications, this work aims at solving the even harder problem of estimating 3D trajectories using 2D laser scanners. These are much cheaper than their 3D counterparts and their reduction in size and weight in recent years allow for truly mobile applications. By actuating such a 2D laser scanner appropriately (fig. 1.1b) it is possible to cover the whole environment visible to the scanner. Reconstructing accurate models from actuated 2D laser scanners however requires specialized algorithms for continuous time trajectory estimation, which are the topic of this thesis.

[1] Product image taken from the manufacturer's website www.faro.com.

## 1.1 Motivation

Due to the fast sensor motion as a result of the actuation, it is in particular necessary to treat the trajectory estimation problem in continuous time. Continuous time estimation theory for application in SLAM has only been fully developed very recently. It allows to move the laser scanner during acquisition by treating scans taken over a timespan as deformable. Additionally, it eases the inclusion of high-rate sensor data such as inertial measurements. Furthermore, it allows to keep the state size to estimate small by representing the trajectory as a sum of few temporal basis functions. The specific choice of trajectory representation is by itself an important part of the estimation problem. It should have local control, high continuity and no singularities when estimating orientations. The Iterative Closest Point (ICP) algorithm was successfully used in 2D SLAM for scan matching to estimate relative pose differences between two rigid 2D laser scans. It can be extended to obtain a general Non-linear Least Squares (NLS) optimization problem for multi-view, continuous-time estimation, which accurately registers scans of an actuated 2D laser scanner in 3D space.

## 1.2 Contributions

The main contributions of this thesis are:

- An extensive evaluation of different trajectory representations.
- An accurate algorithm for continuous time trajectory estimation for 3D SLAM from an actuated 2D laser scanner. All formulas needed for the algorithm are derived.
- An evaluation with ground truth data and different scenarios, noise levels, and parameters.

## 1.3 Outline

The remaining chapters of this thesis are organized as follows:

(chapter 2) A clear treatment of the theoretical background needed for our algorithm. This comprises rigid body motion, sensor hardware, interpolation methods, principal component analysis and non-linear least squares optimization.

(chapter 3) An extensive review of previous work regarding ICP, actuated LiDAR based SLAM, orientation interpolation and continuous time estimation.

(chapter 4) Our main approach is motivated by proposing improvements over existing approaches. A large part is spent on finding a good trajectory representation. After giving an overview over the algorithm, we show how sensor measurements are used and how the employed model can be optimized.

(chapter 5) We evaluate our approach with different simulated scenarios, noise levels and parameters and compare to the available ground truth.

(chapter 6) Summary of achievements and hints for future work.

# 2 Background

This chapter gives the theoretical preliminaries needed to follow the subsequent discussion of previous work as well as the derivation of our approach. We start with the introduction of rigid body motion (section 2.1), which is a crucial prerequisite for 3D motion estimation. Then we describe the used sensor hardware (section 2.2). Continuous time trajectories need to be represented by continuous curves, which is why we cover interpolation techniques next (section 2.3). The basics of 3D shape analysis on point sets is also covered (section 2.4). This chapter is concluded with a general optimization technique that is utilized in our method (section 2.5).

## 2.1 Rigid body motion

A *rigid body motion* $g$ is a family of transformations that describes how the coordinates of an object or point $\mathbf{p} \in \mathbb{R}^3$ in 3D Euclidean space change as a function of time $\tau$. The shape of the object should not change over time: it is solid and not deformable. Thus a rigid body motion must preserve distance and orientation between any pair of points on the object. Each rigid body motion $g$ consists of a translational and a rotational part and has 6 Degree(s) of Freedom (DoF) in total. The translational part is easily minimally parametrized by a displacement vector $\mathbf{t} \in \mathbb{R}^3$, while there exist multiple possible parameterizations for the rotational part.

In the following, we will first revisit different parameterizations of rotations, namely rotation matrices (section 2.1.1) and unit quaternions (section 2.1.2). Then we will talk about two possible parameterizations for the full rigid body motion (section 2.1.3). This section closely follows [MSKS05, Ch. 2], which we refer to for further details.

### 2.1.1 Rotation matrices

#### Special orthogonal group of rotation matrices (SO(3))

A rotation can be represented by a matrix which fulfils special properties. When a vector or point is rotated, it does not change its length. Thus the matrix $\mathrm{R}$ must be an orthonormal matrix. The rotation of a point $\mathbf{p} \in \mathbb{R}^3$ is then done by a simple matrix multiplication $\mathrm{R}\mathbf{p}$. Thus, the rotation matrices $\mathrm{R}$ form SO(3):

$$\boxed{\mathrm{SO}(3) = \{\mathrm{R} \in \mathbb{R}^{3\times 3} \mid \mathrm{R}^T\mathrm{R} = \mathrm{I}, \det(\mathrm{R}) = +1\}}$$

From orthogonality, it easily follows that $\mathrm{R}^{-1} = \mathrm{R}^T$. The rotation matrix $\mathrm{R}$ consists of 9 parameters, even though it only has 3 degrees of freedom (DoF). This means we can only freely choose 3 entries that automatically determine the 6 remaining entries through the conditions $\mathrm{R}\mathrm{R}^T = \mathrm{I}$ and $\det(R) = 1$. This is inconvenient for rotation interpolation and

when estimating rotation parameters in an optimization problem, which is why we will later introduce alternative parameterizations.

### Lie algebra of skew-symmetric 3x3 matrices (so(3))

Given the time ($\tau$) dependent orientation curve of an object $\mathrm{R}(\tau) : \mathbb{R} \to \mathrm{SO}(3)$, it must satisfy $\mathrm{R}(\tau)\mathrm{R}^T(\tau) = \mathrm{I}$. Its time derivative must thus satisfy: $\dot{\mathrm{R}}(\tau)\mathrm{R}^T(\tau) = -(\dot{\mathrm{R}}(\tau)\mathrm{R}^T(\tau))^T$, which means that $\dot{\mathrm{R}}(\tau)\mathrm{R}^T(\tau)$ is a skew-symmetric matrix.

Each skew-symmetric matrix can be uniquely defined by an angular velocity vector $\boldsymbol{\omega} \in \mathbb{R}^3$. The cross product $\times : \mathbb{R}^3 \to \mathbb{R}^3$ has the important property $\boldsymbol{\omega} \times \mathbf{p} = -\mathbf{p} \times \boldsymbol{\omega}$. Since it is a linear map, it can also be represented by a matrix product $\boldsymbol{\omega} \times \mathbf{p} \equiv [\boldsymbol{\omega}]_\times \mathbf{p}$ with the cross product matrix operator $[.]_\times$:

$$[\boldsymbol{\omega}]_\times = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \qquad (2.1)$$
cross product matrix

A skew symmetric matrix has the important property $[\boldsymbol{\omega}]_\times = -[\boldsymbol{\omega}]_\times^T$, in analogy to its derivation via the cross product. $[.]_\times : \mathbb{R}^3 \to \mathfrak{so}(3) \subset \mathbb{R}^{3\times3}$ is an isomorphism between $\mathbb{R}^3$ and the space $\mathfrak{so}(3)$ of all 3x3 skew symmetric matrices, which are the tangent space at the identity of the matrix group $\mathrm{SO}(3)$:

$$\boxed{\mathfrak{so}(3) = \{[\boldsymbol{\omega}]_\times \in \mathbb{R}^{3\times3} \mid \boldsymbol{\omega} \in \mathbb{R}^3\}}$$

### Matrix exponentiation and logarithm

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 instantaneous rotation defined by it can be obtained by the following chain of operations and vice versa:

$$\boldsymbol{\omega} \in \mathbb{R}^3 \underset{[.]_\times^{-1}}{\overset{[.]_\times}{\rightleftharpoons}} [\boldsymbol{\omega}]_\times \in \mathfrak{so}(3) \underset{\log}{\overset{\exp}{\rightleftharpoons}} \mathrm{R} \in \mathrm{SO}(3)$$

The matrix exponential $\exp$ has an exact and efficient closed form Taylor series expansion, the Rodrigues' rotation formula:

$$\exp[\boldsymbol{\omega}]_\times = \exp[\theta\hat{\boldsymbol{\omega}}]_\times = \mathrm{I} + [\hat{\boldsymbol{\omega}}]_\times \sin\theta + [\hat{\boldsymbol{\omega}}]_\times^2 (1 - \cos\theta)$$

The Rodrigues formula is also used to convert from the angle-axis representation of rotations $(\theta, \hat{\boldsymbol{\omega}})$, yet another frequently used method to represent rotations, to a rotation matrix. The above formula is only surjective, but there exists an efficient way to compute the matrix logarithm $\log$, the 'inverse' of above formula [MSKS05, Theorem 2.1].

When using the small angle $|\theta| << 1$ approximation, it holds that $sin(\theta) \approx \theta$ so by approximating $\sin(\theta)$ with $\theta$ and dropping the second and higher order terms from above equation, we gain a first-order Taylor expansion of infinitesimal rotations:

$$\mathrm{R} \approxeq \mathrm{I} + [\boldsymbol{\omega}]_\times \qquad (2.2)$$
infinitesimal rotation

This approximation is linear in $\boldsymbol{\omega}$ and is thus often used when estimating orientations via linear algebra techniques or iteratively in optimization.

### 2.1.2 Quaternions

A *quaternion* $\mathbf{q}$ is basically 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:

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

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

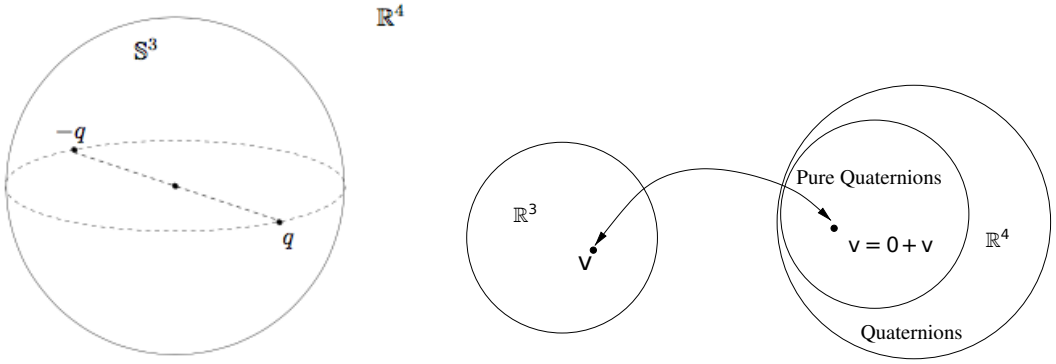
$$\begin{aligned} \text{adjoint/conjugate}: \quad & \bar{\mathbf{q}} = [q_w, -\mathbf{q}_{x:z}]^T \\ \text{norm}: \quad & \|\mathbf{q}\| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} \\ \text{inverse}: \quad & \mathbf{q}^{-1} = \frac{\bar{\mathbf{q}}}{\|\mathbf{q}\|} \end{aligned}$$

Multiplication of two quaternions is associative and distributive, but anti-commutative since $\mathbf{q}\mathbf{q}' \neq \mathbf{q}'\mathbf{q}, ij = -ji$ and defined as [DKL98, Jia08]:

$$\mathbf{q}\mathbf{q}' = [q_w q'_w - \mathbf{q}_{x:z} \cdot \mathbf{q}'_{x:z}, \mathbf{q}_{x:z} \times \mathbf{q}'_{x:z} + q_w \mathbf{q}'_{x:z} + \mathbf{q}_{x:z} q'_w] \tag{2.3}$$

quaternion multiplication

Where $\cdot$ is the dot product and $\times$ the cross product.




(a) **Unit quaternions**: Antipodal unit quaternions $\mathbf{q}$ and $-\mathbf{q}$ on the unit sphere $\mathbb{S}^3 \subset \mathbb{R}^4$ correspond to the same rotation matrix. [MSKS05, Figure 2.1]

(b) **Pure quaternions**: $\mathbb{R}^3$ viewed as the space of pure quaternions. [Jia08, Figure 1]

Figure 2.1: The space of quaternions, its subspaces and their relation to other spaces.

### Special unitary group isomorphic to unit quaternions (SU(2))

Just as complex numbers can be used to represent rotations in $\mathbb{R}^2$ using Euler's formula $e^{i\varphi} = \cos(\varphi) + i\sin(\varphi)$, 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 $\mathbb{S}^3$. The Special unitary group isomorphic to unit quaternions (SU(2)) is:

$$\boxed{\mathrm{SU}(2) \cong \mathbb{S}^3 \cong \mathbb{H}_1 = \{\mathbf{q} \in \mathbb{H} \quad | \quad \|\mathbf{q}\| = 1\}}$$

Note that topologically, the groups $\mathrm{SU}(2) \cong \mathbb{S}^3 \cong \mathbb{H}_1$ are all isomorphic to each other. This allows to transfer topological properties of one representative to the other ones. We can make us of this isomorphism to draw the connection to rotation matrices, because SU(2) is a universal cover of SO(3). Since the kernel of this cover has order 2, this is a double covering of $\mathrm{SO}(3)$, meaning that the two antipodal unit quaternions $\mathbf{q}$ and $-\mathbf{q}$ both represent the same orientation or rotation matrix (fig. 2.1a). Because of this, the distance measure in $\mathbb{S}^3$ is also twice as long as that of $\mathrm{SO}(3)$.

To rotate a vector or point $\mathbf{p} \in \mathbb{R}^3$ by the unit quaternion $\mathbf{q} \in \mathbb{H}$, one must apply:

$$\mathbf{qp} := \mathbf{qp}\bar{\mathbf{q}} \qquad (2.4)$$
quaternion rotation

where the 3D vector $\mathbf{p}$ is interpreted as a pure quaternion $[0, \mathbf{p}]^T$ with the scalar part set to zero, and then the usual quaternion multiplication is applied. Note also that for unit quaternions, it holds that $\mathbf{q}^{-1} = \bar{\mathbf{q}}$. As a convention, we write $\mathbf{qp}$ to express quaternion rotation, not to be confused with quaternion composition $\mathbf{qq}'$, which is the direct application of quaternion multiplication.

### Lie algebra isomorphic to pure quaternions (su(2))

The group of unit quaternions $\mathbb{H}_1$ is isomorphic to the Lie group $\mathrm{SU}(2)$. Thus, its corresponding Lie algebra $\mathfrak{su}(2)$ should also have a representation via quaternions. These are exactly the pure quaternions $\mathbb{H}_0$:

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

Since the scalar part of these quaternions is zero, we can identify their vector part with a vector in $\mathbb{R}^3$ (fig. 2.1b), which can be interpreted as an angular velocity vector [Hol08]. Topologically, the algebras $\mathfrak{su}(2) \cong \mathbb{R}^3 \cong \mathbb{H}_0$ are all isomorphic to each other, so we can choose whichever representative suits us best.

#### Quaternion exponentiation and logarithm

Similar to matrices, one can also define exponentiation and logarithm for quaternions. These functions allow to convert between Lie group and algebra as follows:

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

Note that in contrast to the matrix representation, there is no intermediate representation such as a skew-symmetric matrix needed in this case. However, one has to take care of the double covering and the 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

$$\exp(\boldsymbol{\omega}) = [\cos(0.5\theta), \hat{\boldsymbol{\omega}} \sin(0.5\theta)]^T = [q_w, \mathbf{q}_{x:z}]^T = \mathbf{q} \in \mathbb{S}^3 \tag{2.5}$$
quaternion exp

becomes the unit quaternion which represents the rotation by angle $\theta$ about the axis $\hat{\boldsymbol{\omega}}$ [KKS95]. The exponential map exp can 1) be interpreted as a mapping from the angular velocity vector $\boldsymbol{\omega}$ (measured in SO(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 $2\theta$ instead of $\theta$ around $\hat{\boldsymbol{\omega}}$, because $\boldsymbol{\omega}$ is then measured in $\mathbb{S}^3$ instead of $\mathrm{SO}(3)$.

Since $\cos$ and $\sin$ are periodic functions, an inverse of $\exp$, called $\log$, only exists if we limit the domain of $\theta < 2\pi$:

$$\log(\mathbf{q}) = 2\arccos(q_w)\frac{\mathbf{q}_{x:z}}{\|\mathbf{q}_{x:z}\|}^T = \theta\hat{\boldsymbol{\omega}} = \boldsymbol{\omega} \in \mathbb{R}^3 \tag{2.6}$$
quaternion log

Again, if we omit the scalar constant $2$ in above equation, $\boldsymbol{\omega}$ is measured in $\mathbb{S}^3$ instead of $\mathrm{SO}(3)$.

### 2.1.3 Full rigid body motion parameterizations

#### Special Euclidean group of rigid body motions (SE(3))

One way to represent the full rigid body motion, consisting of translation and rotation, are homogeneous coordinates and $4 \times 4$ matrices, a representation for SE(3):

$$\boxed{\mathrm{SE}(3) = \left\{ \mathrm{T} = \begin{bmatrix} \mathrm{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \;\middle|\; \mathrm{R} \in \mathrm{SO}(3), \mathbf{t} \in \mathbb{R}^3 \right\}}$$

This representation comes in handy when points need to be transformed, since one has to carry out a simple matrix vector multiplication. A point $\mathbf{p} \in \mathbb{R}^3$, given as a column vector $\tilde{\mathbf{p}} = [p_1, p_2, p_3, 1]^T$ in homogeneous coordinates is transformed via $\mathrm{T}\tilde{\mathbf{p}}$. Moreover, multiple transformations can be composed by simple multiplication of their homogeneous $4 \times 4$ matrix representations.

$$\mathrm{T}\mathbf{p} = \mathrm{T}\tilde{\mathbf{p}} = \mathrm{R}\mathbf{p} + \mathbf{t} \tag{2.7}$$
transformation

$$\mathrm{T}^{-1} = \begin{bmatrix} \mathrm{R}^T & -\mathrm{R}^T\mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{2.8}$$
inverse

$$\mathrm{T} \oplus \mathrm{T}' = \mathrm{T}\mathrm{T}' = \begin{bmatrix} \mathrm{R}\mathrm{R}' & \mathrm{R}\mathbf{t}' + \mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{2.9}$$
composition

**Lie algebra of twists (se(3))**

Being a Lie group, SE(3) also has its associated Lie algebra of twists $\hat{\xi}$, which we will shortly define here:

$$\boxed{\mathfrak{se}(3) = \left\{ \hat{\xi} = \begin{bmatrix} [\boldsymbol{\omega}]_\times & \mathbf{v} \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{4\times4} \;\middle|\; [\boldsymbol{\omega}]_\times \in \mathfrak{so}(3), \mathbf{v} \in \mathbb{R}^3 \right\} \subset \mathbb{R}^{4\times4}}$$

Its interesting property is that the vectors $\boldsymbol{\omega}$ and $\mathbf{v}$ can be interpreted as angular and linear velocity. There also exist versions of SE(3) exponentiation and SE(3) logarithm (4.5) to convert between algebra and group, which will be analyzed later on (section 4.2.4).

**Full rigid body motion with quaternions**

Another way to parameterize SE(3) doesn't use matrix-vector products to express rotation by $\mathrm{R}$ and homogeneous coordinates to further express addition of $\mathbf{t}$, but quaternion multiplication and standard vector addition.

$$(\mathbf{q}, \mathbf{t})\mathbf{p} = \mathbf{q}\mathbf{p} + \mathbf{t} \tag{2.10}$$
transformation

$$(\mathbf{q}, \mathbf{t})^{-1} = (\bar{\mathbf{q}}, -\bar{\mathbf{q}}\mathbf{t}) \tag{2.11}$$
inverse

$$(\mathbf{q}, \mathbf{t}) \oplus (\mathbf{q}', \mathbf{t}') = (\mathbf{q}\mathbf{q}', \mathbf{q}\mathbf{t}' + \mathbf{t}) \tag{2.12}$$
composition

We will mostly be using this representation, since it is singularity free and has at the same time much fewer redundancy and constraints than the matrix representation, only the unity norm constraint is left and one coefficient more than DoF. During optimization, we will locally parameterize updates to a unit quaternion with a pure quaternion, and then map back via quaternion exponentiation. This allows us to do unconstrained optimization without having singularities.

## 2.2 Sensors

### 2.2.1 Light Detection And Ranging (LiDAR)


(a) SICK LMS-291 (b) SICK LMS-100 (c) Hokuyo UTM-30LX

Figure 2.2: commercial 2D laser scanners.[1]

**Li**ght **D**etection **A**nd **R**anging (LiDAR) was originally a field surveying or remote sensing method that measures distance to a target by illuminating that target with a pulsed laser beam. It is sometimes called laser scanning. In recent years 2D laser scanners have successfully been adopted in robotics. The continuing reduction in weight, size and price are appealing reasons to use these devices for trajectory estimation and reconstruction at great precision. The most prominent devices used in the robotics domain are shown in (fig. 2.2). The devices are ordered chronologically. When looking at their specifications, note the drop in weight and size while at the same time the resolution and number of measurements per second increases (table 2.1).

| | weight | height | diameter | frequency | resolution | field of view | pts/scan | pts/s |
|---|---|---|---|---|---|---|---|---|
| SICK LMS-291 | 4,5kg | 21cm | 15cm | 75Hz | 1,00° | 180° | 180 | 13500 |
| SICK LMS-100 | 1,1kg | 15cm | 10cm | 50Hz | 0,50° | 270° | 540 | 27000 |
| Hokuyo UTM-30LX | 0,3kg | 9cm | 6cm | 40Hz | 0,25° | 270° | 1080 | 43200 |

Table 2.1: 2D laser scanner specifications.[2]

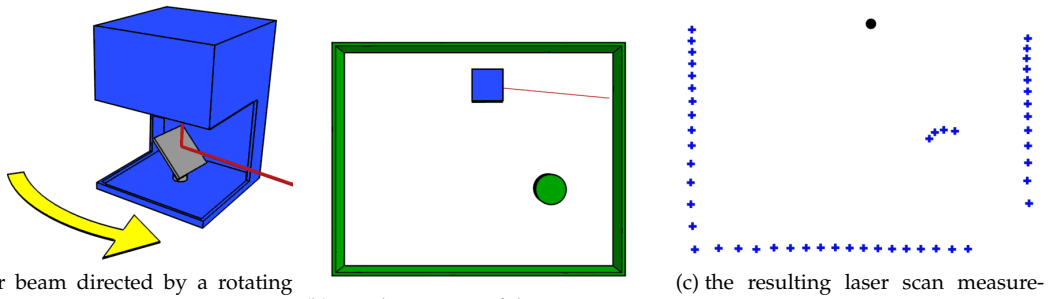
#### Geometry and unprojection

2D laser scanners typically use a mirror rotating around the vertical axis to send the laser beam out at a specific angle (fig. 2.3). The result is an array of range measurements in polar coordinates, where each entry denotes the distance to the closest surface.

The process by which the real-world 2D points $(x_i, y_i)$, measured in the horizontal scan plane of the laser scanner, are mapped into the array or range measurements $r_i, 0 \leq i \leq n$

[1] Product images from the manufacturers websites www.sick.de and www.hokuyo-aut.jp.

[2] Specifications according to the data sheets on the manufacturers websites.



(a) Laser beam directed by a rotating mirror

(b) top-down view of the environment

(c) the resulting laser scan measurement

Figure 2.3: Laser scan technology.[3]

can easily be modeled by a transformation from the Cartesian coordinate system $(x, y)$ into the Polar coordinate system:

$$\begin{pmatrix} \phi \\ r \end{pmatrix} = \begin{pmatrix} \tan^{-1} \frac{y}{x} \\ \sqrt{x^2 + y^2} \end{pmatrix}, i = (\phi - \phi_0)/\Delta\phi$$

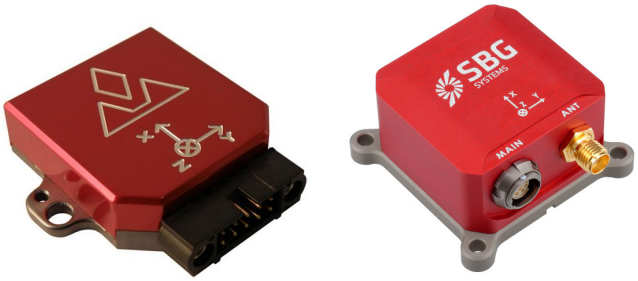
For 2D laser scanners which take measurements at equal angle increments $\Delta\phi$, the angle $\phi$ is absorbed into the array index $i$, given as: $i = (\phi - \phi_0)/\Delta\phi$. The angle increment $\Delta\phi$ together with the starting angle $\phi_0$ is sometimes called the intrinsic parameters of the laser scanner, which also define its field of view.

If we know the intrinsic parameters of the laser scanner, the above process can easily be reversed. This is a necessary conversion, since we will later be dealing with points in Cartesian coordinates only. So to convert a raw measurement $r$ at array index $i$ into a 2D Cartesian coordinate, with the laser scanner at the origin $(0, 0)$, we apply:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r\cos(\phi) \\ r\sin(\phi) \end{pmatrix}, \phi = \phi_0 + i\Delta\phi \tag{2.13}$$

lidar backprojection

### 2.2.2 Inertial Measurement Unit (IMU)




(a) Vectornav VN-100

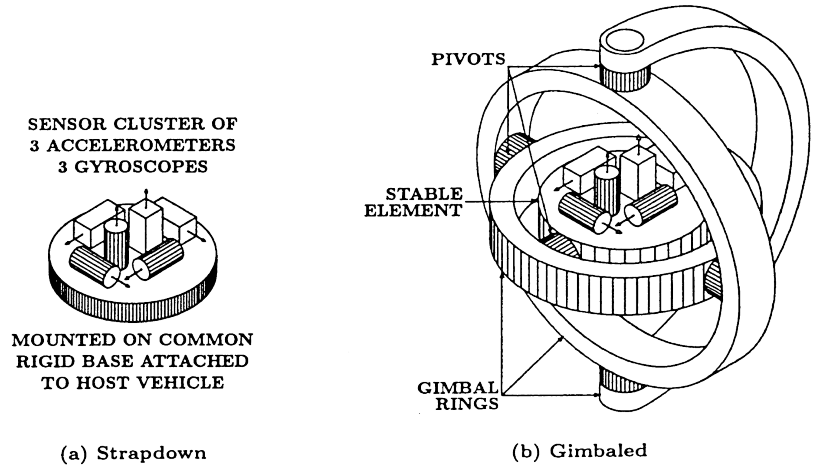
(b) SBG Ellipse-N

Figure 2.4: Commercial strapdown IMU's.[4]

[3] Drawings from https://en.wikipedia.org/wiki/Lidar.

[4] Product images from the manufacturers websites www.vectornav.com and www.sbg-systems.com.

An Inertial Measurement Unit (IMU) is a device that measures the angular rate and linear acceleration of a body. It was originally developed for navigation of ships, airplanes, satellites and missiles. It usually contains 3 accelerometers and 3 gyroscopes for each direction, mounted to a common base and can be implemented (fig. 2.5) in two ways [TW04]:




(a) Strapdown (b) Gimbaled

Figure 2.5: Types of IMU's [GWA07, Fig 2.1]

- Strapdown systems: The inertial sensors are directly connected to the case. Requires large computational power for correct integration.
- Gimbaled or stable platform systems: The inertial sensors are isolated from external rotational motion. Requires heavy mechanic.

Nowadays, strap-down IMUs are the prevalent type for most applications due to their low cost, light weight, and today's processing power (fig. 2.4).

### Sensor frame and noise model

As said before an IMU measures two quantities, the rotation rate or angular velocity $\tilde{\boldsymbol{\omega}} \in \mathbb{R}^3$, measured in $\mathrm{SO}(3)$ and the linear acceleration $\tilde{\mathbf{a}} \in \mathbb{R}^3$. Let us introduce 2 frames, the world frame ${}_w$ and the sensor frame ${}_s$. In strap-down IMU's, these vector-valued quantities are measured in ${}_s$. Both measurements are affected by zero-mean additive white noise as well as subject to slowly varying biases ${}_s\mathbf{b}_{\boldsymbol{\omega}}$ and ${}_s\mathbf{b}_{\mathbf{a}}$. Additionally, the world's gravity ${}_w\mathbf{g}$, rotated into the sensor frame, is contained in the measured acceleration vector:

$$ {}_s\tilde{\boldsymbol{\omega}}(\tau) = {}_s\boldsymbol{\omega}(\tau) + {}_s\mathbf{b}_{\boldsymbol{\omega}}(\tau) + \mathcal{N}(\mathbf{0}, \sigma_{\boldsymbol{\omega}}) \tag{2.14} $$

angular velocity

$$ {}_s\tilde{\mathbf{a}}(\tau) = {}_s\mathbf{a}(\tau) + \bar{\mathbf{q}}(\tau){}_w\mathbf{g} + {}_s\mathbf{b}_{\mathbf{a}}(\tau) + \mathcal{N}(\mathbf{0}, \sigma_{\mathbf{a}}) \tag{2.15} $$

linear acceleration

## 2.3 Interpolating and approximating curves

Given a sequence of $n+1$ control points $\mathbf{p}_0, ..., \mathbf{p}_n \in \mathbb{R}^x$ at discrete times, the question is how to fill in the intermediate values to get a continuous spatial curve. Before choosing an appropriate technique, we must consider multiple issues[Par12, Ch. 3]:

1. Interpolation vs. approximation: An interpolating curve passes through the control points, whereas in approximation, the points merely control the shape of the curve, which doesn't have to pass through them.

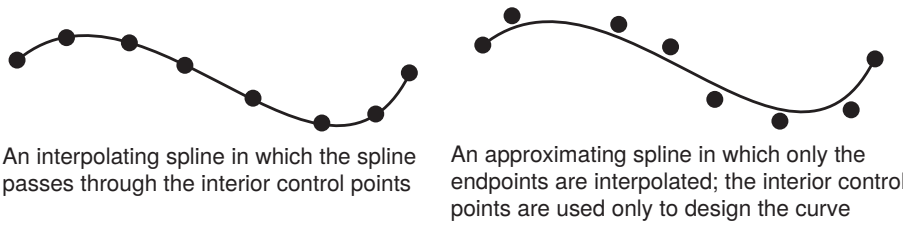



Figure 2.6: Interpolation vs. approximation [Par12, Fig 3.1]

2. Continuity: The smoothness of a curve is determined by how often one can derive it without loosing continuity.
   - $C^0$ or positional continuity means the curve itself has no jumps
   - $C^1$ or tangential continuity is usually sufficient in animation applications, requiring a smooth velocity curve
   - $C^2$ or curvature continuity becomes important when dealing with time-dependent curves, requiring a smooth acceleration curve

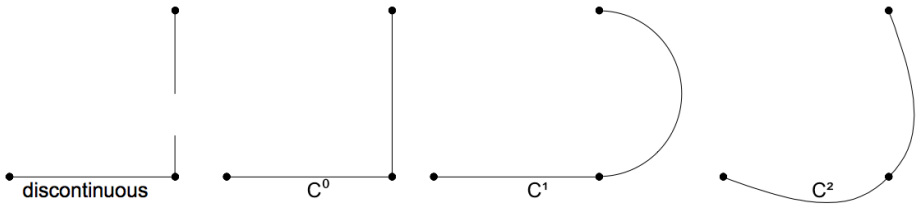



Figure 2.7: Continuity [DKL98, Fig 6.12]

3. Global vs. local control: If a change in a single control point only affects the shape of the curve in a limited region, the curve is said to have local control. If it affects the entire curve, even slightly, it is said to have global control.

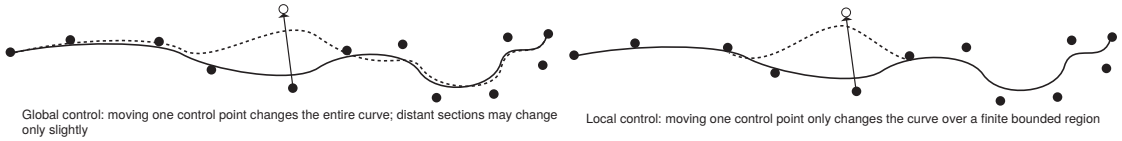



Figure 2.8: Global vs. local control [Par12, Fig 3.3]

4. Complexity: computational effort needed to construct and evaluate the curve.

### 2.3.1 Linear interpolation (LERP)

The line connecting $\mathbf{p}_0, \mathbf{p}_1 \in \mathbb{R}^x$ is given in $u \in [0,1]$ by **l**inear int**erp**olation [Han06, 25.1]:

$$lerp(\mathbf{p}_0, \mathbf{p}_1, u) \quad = \quad (1-u)\mathbf{p}_0 + u\mathbf{p}_1 \quad = \quad \mathbf{p}_0 + u(\mathbf{p}_1 - \mathbf{p}_0) \tag{2.16}$$
LERP

If more than 2 points need to be interpolated, the easiest way is to use piecewise linear interpolation, generating a line segment for each consecutive pair of points. Piecewise linear interpolation curves interpolate all points, have local control and low complexity, but are only $C^0$ 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, which is why higher order interpolation methods have been developed.

### 2.3.2 Bézier curve

To gain higher order continuity, one can apply LERP (2.16) to pairs of $n+1$ consecutive control points in an iterative manner. This geometric construction scheme by *repeated linear interpolation* is known as the **de Casteljau** algorithm [Far02, Ch. 4], defined by the following recurrence relation:

$$\begin{aligned}\mathbf{p}_i^{(0)} &= \mathbf{p}_i \quad i = 0, ..., n-k, \quad k = 1, ..., n \\ \mathbf{p}_i^{(k)} &= (1-u)\mathbf{p}_i^{(k-1)} + u\mathbf{p}_{i+1}^{(k-1)}\end{aligned}$$

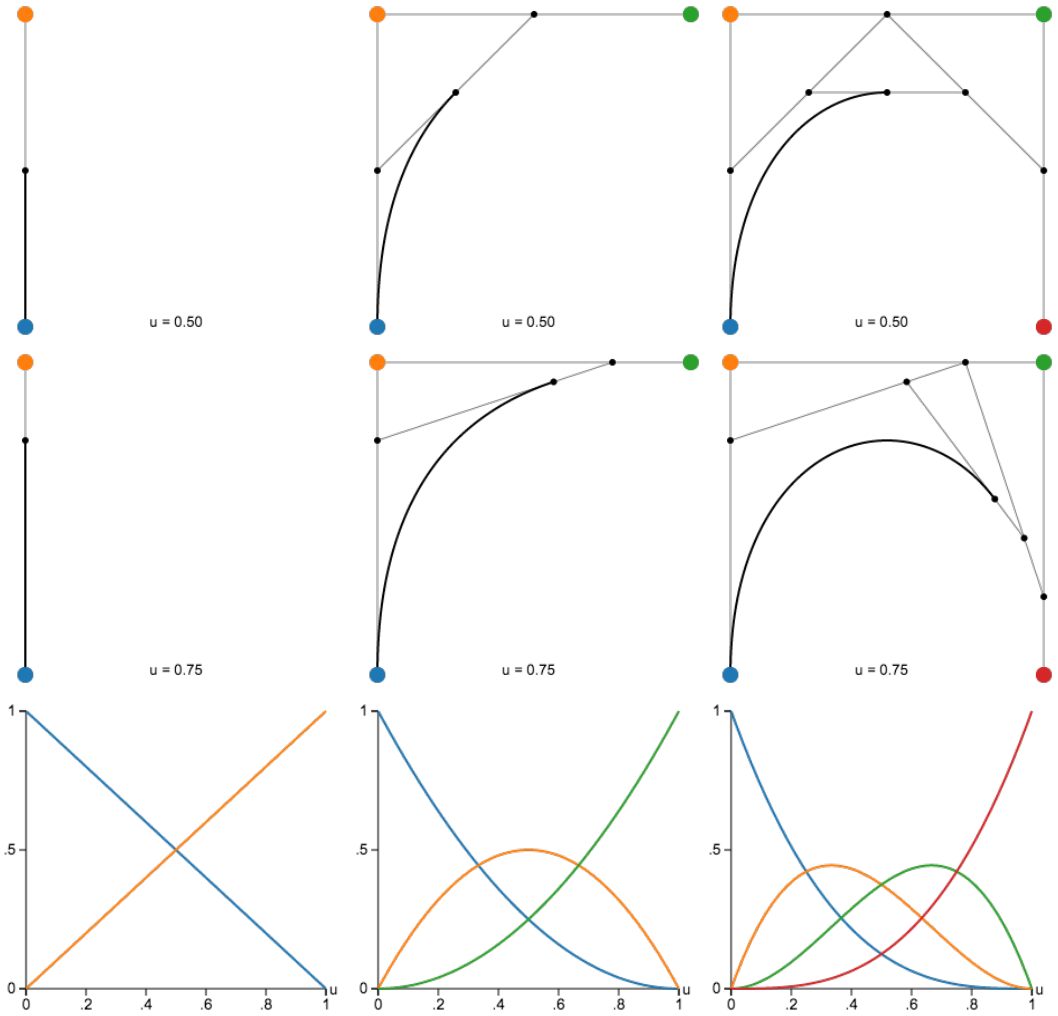
It allows recursively computing a **Bézier curve** [Far02, Ch.5] [Mor97, Ch. 4] [Par12, Appendix B.5.9] of degree $n$, for example (fig. 2.9) :

$$\begin{aligned}n = 1 : \mathbf{p}(u) &= (1-u)\mathbf{p}_0 + u\mathbf{p}_1 \\ n = 2 : \mathbf{p}(u) &= (1-u)\mathbf{p}_0^{(1)} + u\mathbf{p}_1^{(1)} = (1-u)\big((1-u)\mathbf{p}_0 + u\mathbf{p}_1\big) + u\big((1-u)\mathbf{p}_1 + u\mathbf{p}_2\big) \\ &= (1-u)^2\mathbf{p}_0 + 2u(1-u)\mathbf{p}_1 + u^2\mathbf{p}_2 \\ n = 3 : \mathbf{p}(u) &= (1-u)\mathbf{p}_0^{(2)} + u\mathbf{p}_1^{(2)} = \dots \\ &= (1-u)^3\mathbf{p}_0 + 3u(1-u)^2\mathbf{p}_1 + 3u^2(1-u)\mathbf{p}_2 + u^3\mathbf{p}_3\end{aligned}$$

The weights of the control points $\mathbf{p}_i$ are exactly the **Bernstein polynomials** $B_i^n(u)$, which allow for an explicit representation of a Bézier curve of a certain degree $n$ [Far02, Ch. 5]:

$$\mathbf{p}(u) = \sum_{i=0}^{n} B_i^n(u)\mathbf{p}_i \quad , \quad B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i} \tag{2.17}$$
Bézier curve

The above curve can also be written in **matrix form**. For the cubic Bézier curve ($n = 3$),




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

Figure 2.9: Bézier curves of degree 1,2 and 3 evaluated at different times using the de Casteljau algorithm. The control points are associated by colour with their respective weights in the interval $u \in [0, 1]$, given by the Bernstein polynomials of the corresponding degree.

defined by $4$ control points, we get [Mor97, 84][Par12, 460]:

$$\mathbf{p}(u) = [u^3, u^2, u, 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \tag{2.18}$$

Bézier matrix

A Bézier curve interpolates the first and last control point, while the inner ones are approximated. Being a polynomial, a Bézier curve is infinitely many times 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.

### 2.3.3 B-spline

A **B-spline** [Far02, Ch. 8] [Mor97, Ch. 5] [Par12, Appendix B.5.12] is a piecewise polynomial function and a generalization of a **Bézier curve**. It is defined over a **knot sequence** $\{\tau_i\}, \tau_i \leq \tau_{i+1}$, which allows to decouple the degree $k-1$ or order $k$ of the curve from the number $n+1$ of control points, in contrast to the **Bézier curve**. Each control point $\mathbf{p}_i$ is associated with its knot $\tau_i$. The B-spline curve $\mathbf{p}(\tau)$ is defined as a weighted sum of B-spline basis functions $N_i^k$:

$$\mathbf{p}(\tau) = \sum_{i=0}^{n} N_i^k(\tau)\mathbf{p}_i \tag{2.19}$$
B-spline

Where the basis functions themselves adhere to the following recurrence, which can be computed using the **de Boor** algorithm:

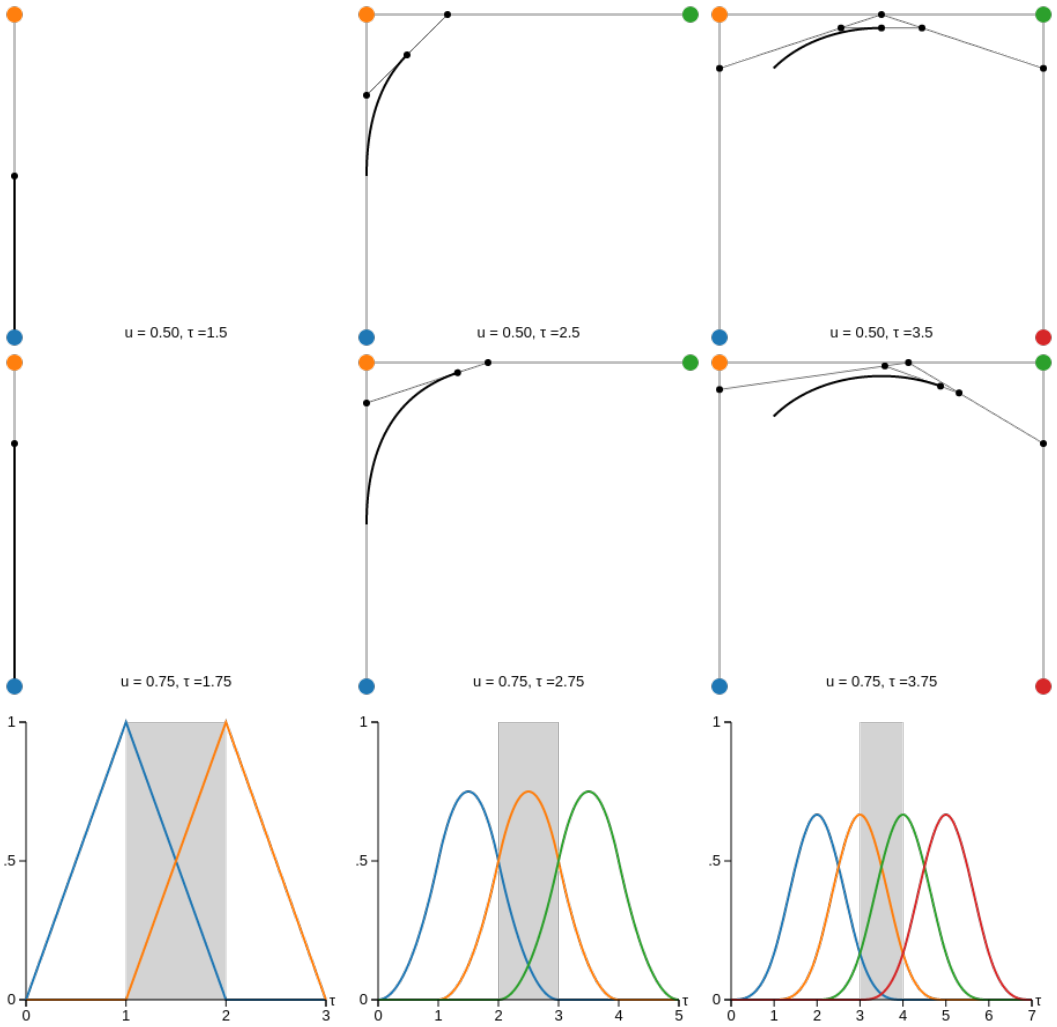
$$\begin{aligned} N_i^0(\tau) &= \begin{cases} 1 & \text{if } \tau \in [\tau_i, \tau_{i+1}) \\ 0 & \text{otherwise} \end{cases} \\ N_i^k(\tau) &= (1-\alpha_{i,k})N_{i-1}^{k-1}(\tau) + \alpha_{i,k}N_i^{k-1}(\tau) \\ \alpha_{i,k} &= \frac{\tau - \tau_i}{\tau_{i+n+1-k} - \tau_i} \end{aligned}$$

The above formula looks similar to the Bézier curve, but instead of blending basis functions iteratively with weights $(1-u)$ and $u$, these are now replaced by $1-\alpha_{i,k}$ and $\alpha_{i,k}$, which change at each iteration. A new global time variable $\tau \in [\tau_0, ..., \tau_{n+1}]$ is introduced while the parametric variable $u \in [0, 1]$ is not necessary anymore, but can be used as a local variable for a B-spline segment between two knots (fig. 2.10).

In practice, we are mostly interested in $C^2$ continuous curves, and thus need to use at least cubic (degree=3, order=4) B-splines. A uniform knot spacing furthermore simplifies computation and notation while still being able to represent most curves. For a uniform cubic B-spline, we can give a **matrix form** as follows [Mor97, 125][Par12, 464]:

$$\mathbf{p}(\tau) = [u^3, u^2, u, 1]\frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \end{bmatrix}, u = \frac{\tau - \tau_i}{\tau_{i+1} - \tau_i}, \tau \in [\tau_i, \tau_{i+1}) \tag{2.20}$$
B-spline matrix

A B-spline has local support. 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 $\tau$, 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 in the uniform knot spacing case, **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.




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

Figure 2.10: B-spline segment of degree 1,2 and 3 evaluated with the de Boor algorithm. The gray area indicates the current segment $u \in [0, 1]$ between the two knots $\tau_i, \tau_{i+1}$. 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.

## 2.4 Principal Component Analysis (PCA)

In the following we describe the Principal Component Analysis (PCA), a mathematical tool that allows to compute surface statistics such as curvature (2.24) and normal (2.25) of a point set. Later in our method we will make use of this tool to process point measurements acquired by laser scanners (section 4.5.2).

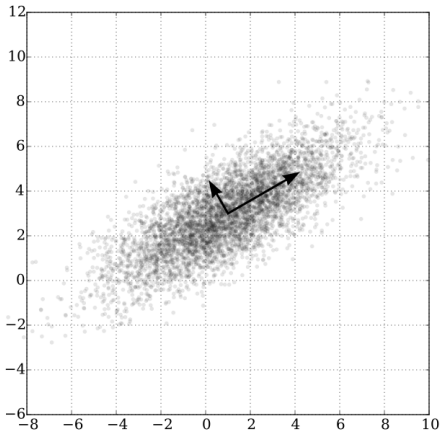



Figure 2.11: PCA of a multivariate Gaussian (2D) distribution. The mean is at (1,3). The two eigenvectors of the covariance matrix are scaled by their corresponding eigenvalues with a standard deviation of 3 in the (0.866, 0.5) direction and of 1 in the orthogonal direction.[5]

To supply a point $\mathbf{p} \in \mathbb{R}^3$ with a surface normal $\vec{\mathbf{n}}$, we first try to fit a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ to all the $n$ points $\{\mathbf{p}_i\}$ in the vicinity of the measurement point. The mean or centroid $\boldsymbol{\mu}$ and the covariance matrix $\Sigma$ are computed as follows:

$$\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{p}_i \in \mathbb{R}^3 \qquad (2.21)$$

centroid

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{p}_i - \boldsymbol{\mu})(\mathbf{p}_i - \boldsymbol{\mu})^T \in \mathbb{R}^{3\times 3} \qquad (2.22)$$

covariance matrix

The 9 entries of the covariance matrix not only describe how much the points deviate from their centroid, but also the amount of correlation between the different dimensions.

Principal Component Analysis (PCA) is a statistical method which converts a set of correlated variables into a set of linearly uncorrelated ones, using an orthogonal basis transformation. This is done by eigen decomposition of the covariance matrix $\Sigma$:

$$\Sigma = \mathrm{UDU}^T, \mathrm{U} = \begin{bmatrix} | & | & | \\ \mathbf{e}_0 & \mathbf{e}_1 & \mathbf{e}_2 \\ | & | & | \end{bmatrix}, \mathrm{D} = \begin{bmatrix} \lambda_0 & & \\ & \lambda_1 & \\ & & \lambda_2 \end{bmatrix} \qquad (2.23)$$

eigen decomposition

[5] Plot from https://en.wikipedia.org/wiki/Principal_component_analysis.

The eigenvectors $\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2 \in \mathbb{R}^3$ of the covariance matrix are called principal components and point in the direction of largest variance, measured by their associated eigenvalues. Being the eigenvectors of the symmetric covariance matrix, the principal components are orthogonal and thus uncorrelated to each other. The transformation is defined in such a way that the principal components are ordered by the amount of variance in their direction, i.e. the magnitude of their corresponding eigenvalues $\lambda_0 \leq \lambda_1 \leq \lambda_2 \in \mathbb{R}$.

### 2.4.1 Curvature estimation

PCA can be used to get an estimate of the mean curvature of the underlying surface. In [PGK02, 2.1: Covariance Analysis], the authors approximate the mean curvature at a point $\mathbf{p}$ with what they call the surface variation $\sigma \in [0, \frac{1}{3}]$ based on $n$ neighbours. It is defined as the ratio of $\lambda_0$ which quantitatively describes the variation along the surface normal to the sum of all eigenvalues. A value of $\sigma = 0$ indicates a flat region, while the maximum value $\sigma = 1/3$ is attained for totally isotropically distributed points:

$$\sigma = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{2.24}$$
curvature

### 2.4.2 Normal estimation

If the curvature indicates a flat region, it makes sense to compute the normal of that local plane. The two principal components $\mathbf{e}_1, \mathbf{e}_2$ associated with the larger two eigenvalues $\mathbf{e}_1, \mathbf{e}_2$ span the plane tangent to the point, while the other principal component, $\mathbf{e}_0$ is the normal of that plane. This is due to the fact that its corresponding eigenvalue $\lambda_0$ is the smallest of all three, since the variation of points approximately lying in a plane is smallest in the direction of the plane normal. For every tangent plane, there are two possible normals which point in opposite directions. If the points were acquired by a LiDAR scanner and given in its sensor frame with the scanner at the origin, we can align the normals by checking the angle between the normal and the vector from the scanner origin to the centroid $\boldsymbol{\mu}$. One needs to flip all normals whose dot product with the centroid is smaller than zero, meaning that they were pointing more than 90 degrees away from each other. Furthermore, since normals only specify a direction, their magnitude is irrelevant. Thus, they must be normalized to be unique:

$$\vec{\mathbf{n}} = \frac{\mathbf{e}_0}{\|\mathbf{e}_0\|} sign(\boldsymbol{\mu}^T \mathbf{e}_0) \tag{2.25}$$
normal

Having unique normals all of unit length allows to compare two of them approximately in Euclidean space by looking at their coefficient differences. This has the advantage that we can treat them like usual vectors in $\mathbb{R}^3$ and apply similar techniques for nearest neighbor queries, e.g. storing them in a kd-tree for fast lookups, which will be used later on in our method.

## 2.5 Non-linear Least Squares (NLS)

We now turn to the Non-linear Least Squares (NLS) problem and optimization algorithms for its solution. These kind of problems come up in many areas, e.g curve fitting in statistics or minimizing re-projection errors in computer vision. It is very relevant for this thesis because the solution of such a problem is at the core of our approach (section 4.6).

The goal for the problem's solution is to find a model function that minimizes a sum of squared residuals, that is the sum of squared differences between observed and modeled values. While the model function to optimize in *linear* least squares problems is a linear combination of the parameters to estimate that can always be written in matrix form $Ax = b$ and solved efficiently via various linear algebra techniques, this is not possible for *non-linear* least squares problems where the parameters appear as functions $f_i$. The general form of a NLS problem is:

$$E(x) = \sum_{i}^{N} \left\| f_i\left(x_{i_1}, ..., x_{i_j}\right) \right\|^2 = \sum_{i=1}^{N} \mathbf{r}_i^T \mathbf{r}_i = \mathbf{r}^T \mathbf{r} \tag{2.26}$$
NLS

The $\left[x_{i_1}, ..., x_{i_j}\right]$ are the $j$ parameter blocks of the $i$'th cost function, whose current values define the $i$'th residuum $\mathbf{r}_i = f_i\left(x_{i_1}, ..., x_{i_j}\right)$. By stacking all of the $N$ $\mathbf{r}_i$'s into $\mathbf{r}$, the vector of concatenated residuals, the summation of squared L2 vector norms $||.||^2$ can be rewritten compactly using the dot product.

The Levenberg-Marquardt (LM) algorithm is an optimization procedure which solves NLS (2.26) iteratively. It combines gradient descent and Gauss-Newton approaches to function minimization. The goal at each iteration $k$ is to choose an update $\Delta x$ to the current estimate $x_k$, so that the new estimate $x_{k+1} = x_k + \Delta x$ reduces the error $E(x)$. The idea is to approximate the residuals $\mathbf{r}_i$ by their first order Taylor series expansion:

$$\mathbf{r}_i(x + \Delta x) \approx \mathbf{r}_i(x) + \nabla \mathbf{r}_i(x)\Delta x = \mathbf{r}_i + J_i \Delta x \tag{2.27}$$

Here, $J_i = \frac{\partial \mathbf{r}_i}{\partial x} = \nabla \mathbf{r}_i(x)$ is the Jacobian of $\mathbf{r}_i$, computed in $x$. Similarly to stacking the residuals $\mathbf{r}_i$ into $\mathbf{r}$, we can stack the Jacobians $J_i$ into $J = \frac{\partial \mathbf{r}}{\partial x} = \nabla \mathbf{r}(x)$. With the approximation (2.27), the sum of squared residuals in NLS (2.26) becomes:

$$\begin{aligned} E(x + \Delta x) &= \mathbf{r}(x + \Delta x)^T \mathbf{r}(x + \Delta x) \\ &\approx (\mathbf{r} + J\Delta x)^T (\mathbf{r} + J\Delta x) \\ &= \mathbf{r}^T \mathbf{r} + \mathbf{r}^T J \Delta x + (J\Delta x)^T \mathbf{r} + (J\Delta x)^T (J\Delta x) \\ &= \mathbf{r}^T + 2\Delta x^T J^T \mathbf{r} + \Delta x^T J^T J \Delta x \end{aligned}$$

At each iteration, the task is to find an update step $\Delta x$ which will minimize $E(x + \Delta x)$. Differentiating the above expression with respect to $x$ and equate with zero gives:

$$\nabla_x E(x + \Delta x) = \underbrace{2J^T \mathbf{r}}_{b} + \underbrace{2J^T J}_{H} \Delta x = 0$$

This is an expression involving $b$, the gradient, and $H$, the Gauss-Newton approxima-

tion to the Hessian. The useful property of this approximation of the Hessian is that it only requires first order derivatives, not second order ones like the true Hessian. It is thus much easier to derive analytically or compute numerically. Solving the linear system $Hx = -b$ for x yields the Gauss-Newton update $\Delta x = -(J^T J)^{-1} J^T \mathbf{r}$. Whether this step actually reduces the error $E$ depends on the accuracy of the Taylor series expansion at $x$ and the validity of the Gauss-Newton approximation. This is usually the case when near the minimum. On the other hand, a simple gradient descent step $\Delta x = -\lambda^{-1} J^T \mathbf{r}$ guarantees to reduce $E$, provided that $\lambda$ is sufficiently large (and thus the step size $\lambda^{-1}$ small), but its convergence near the minimum is very slow. The LM algorithm combines both approaches in quite a simple way:

$$\Delta x = -(J^T J + \lambda I)^{-1} J^T \mathbf{r} \tag{2.28}$$

Levenberg step

Large $\lambda$ correspond to small, safe gradient descent steps, which are sure to decrease the error, while small $\lambda$ allow fast convergence near the minimum, but might otherwise increase the error. The Levenberg step (2.28) can also be obtained by minimizing the damped version, called the augmented normal equations $(H + \lambda I)\Delta x = -b$ instead of $H\Delta x = -b$. The scaled identity matrix summand $\lambda I$ acts like a regularizer, meaning that, even if $H$ is close to singular, Cholesky decomposition of $H + \lambda I$ to solve the above would still work. The art of a good LM implementation is to tune $\lambda$ after each iteration to allow for fast convergence: If the new error is lower than the previous one, the update $\Delta x$ is accepted and $\lambda$ is decreased. If the new error is larger than the previous one, the update $\Delta x$ is discarded and $\lambda$ is increased.

The LM algorithm is a *Trust Region* method, because it locally approximates the objective function with a quadratic model function that can be solved linearly for the update step. It is a restricted update step because the *Trust Region* is either expanded or contracted, depending how good the model fits the objective function, by tuning $\Delta x$. Google's Ceres Solver [AM] implements *Line Search* as well as *Trust Region* algorithms to solve NLS (2.26) iteratively. It allows to model a NLS (2.26) problem independent of the algorithm used to solve it. The implemented *Line Search* algorithms, such as steepest descent, and *Trust Region* algorithms, such as LM or Powell's Dogleg, can be switched out seamlessly.

# 3 Related work

With the background acquired in the last chapter, we are now ready to discuss the previous work related to our approach. In general, one can subdivide the contributions to our method into 4 categories. The first discusses the classical algorithm for point cloud registration (section 3.1) and important variants of it. The next served as main inspiration for this work and considers the hardware and algorithm design to make full 3D SLAM from actuated 2D LiDAR scanners possible (section 3.2). We then go back 20 years into the past by looking at classical methods to interpolate in the space of orientations (section 3.3) from a computer graphics perspective. Finally we return to the present by revisiting the recent progress made in continuous time estimation (section 3.4) in the fields of computer vision and robotics.

## 3.1 Iterative Closest Point (ICP)

The optimal registration of point clouds acquired e.g. by LiDAR is an important subproblem in SLAM. It allows building coherent maps from multiple measurements and localize within them. Given the points from two scans taken at different locations and named data and model point cloud, we wish to estimate the relative rigid body transformation $g = (\mathrm{R}, \mathbf{t})$ that aligns them. However, the problem is that the true correspondences between the individual points needed to align them are not known beforehand.

The key idea of the ICP algorithm, introduced by [BM92, CM91], is to use closest point correspondences between data and model point clouds to approximate the true correspondences. It can be summarized in two steps, which are iteratively repeated until convergence to the desired solution is achieved:

I Compute **correspondences** $\{\mathbf{p}_i \to \mathbf{p}'_i\}_1^N$ between two scans: for each data point $g(\mathbf{p}_i)$ transformed by the current estimate $g$, find its closest model point $\mathbf{p}'_i$.

II Update the current **transformation** estimate $g$ so that it minimizes an error metric defined on these correspondences, i.e. the distance between corresponding points.

Since both steps must reduce the error, convergence to a local minimum is guaranteed [Fit03]. Furthermore, an easy termination criterion is met if the set of correspondences from step I does not change compared to the previous iteration. This means the previous transformation in II already brought the energy to a local minimum, so the current transformation would be the identity.

### 3.1.1 Taxonomy

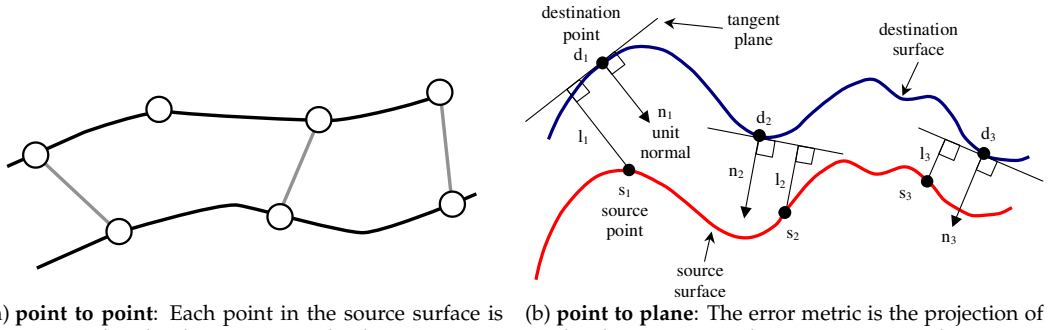
ICP is a popular and well-studied algorithm for 3D shape alignment. Over the years, there have been various modifications to the original algorithm, resulting in a plethora of

different, problem-specific variants. The detailed study [RL01] establishes a taxonomy that allows to categorize and compare these different variants in a unified manner:

1. **Selection** of some set of points in one or both input point sets.
2. **Matching** these points to compatible points in the other set.
3. **Weighting** the corresponding pairs appropriately.
4. **Rejecting** certain pairs as outliers.
5. **Error metric** definition on the set of inliers.
6. **Minimization** of the error metric.

### 3.1.2 Variants

In the following we will first present the two classical and most well-known ICP variants. In recent years, ICP has seen numerous extensions. We especially focus on those variants relevant in the context of our proposed approach (section 4.5.3). Of particular interest is the choice of error metric (5.), which furthermore limits the way minimization (6.) can be carried out.




(a) **point to point**: Each point in the source surface is registered to the closest point in the destination surface. The error metric is the Euclidean distance between these point correspondences.

(b) **point to plane**: The error metric is the projection of the distance vector between corresponding points $\mathbf{s_i} - \mathbf{d_i}$ onto the surface normal of the destination surface $\mathbf{n_i}$, giving the projected vector $\mathbf{l_i}$. [Low04]

Figure 3.1: Error metrics of classical ICP variants. The goal of both is to register each point of the source surface (lower curve) with the destination surface (upper curve). Correspondences are shown as straight lines between the two curves.

#### Point to point

The original and simplest **error metric** (5.) introduced by [BM92] penalizes the Euclidean distance (fig. 3.1a) between a data point $\mathbf{p}_i$, transformed by the current estimate, and its model point $\mathbf{p}'_i$:

$$E = \sum_{i=1}^{N} \left\| \mathrm{R}\mathbf{p}_i + \mathbf{t} - \mathbf{p}'_i \right\|^2 \qquad (3.1)$$
point to point

**Minimization** (6.) is usually done via a closed form solution e.g. using the Singular Value Decomposition (SVD) approach [AHB87]. For more details see [Haa15, 3.6.2, A.2].

#### Point to plane

Introduced by [CM91], who considered the more specific problem of aligning range data for object modeling. They take advantage of the fact that most range data is locally planar and can thus be supplied with surface normals $\vec{\mathbf{n}}'_i$, describing the tangent plane for each model point $\mathbf{p}'_i$ (fig. 3.1b). The **error metric** (5.) is the distance between a data point to the tangent plane of its model point, implemented as the projection of the Euclidean distance between the points, as in the point to point error, onto the normal of the model point. This lets flat regions slide along each other, which turns out to be very advantageous for the alignment of point sampled surfaces from range scans: The location of points in two scans belonging to the same surface may be different due to sampling, but their tangent planes and thus normals are still the same:

$$E = \sum_{i=1}^{N} (\mathrm{R}\mathbf{p}_i + \mathbf{t} - \mathbf{p}'_i) \cdot \vec{\mathbf{n}}'_i \qquad (3.2)$$

point to plane

**Minimization** (6.) is usually done via an approximation by locally linearizing the rotation matrix (2.2). Then, a closed form solution can be obtained from the overdetermined set of equations of the approximation, a standard linear least-squares problem of the form Ax = b. This can be solved e.g. using the pseudo-inverse [Low04], or by Cholesky decomposition of the positive semidefinite matrix obtained from the normal equations. For more details see [Haa15, 3.6.3, A.3].

#### Levenberg-Marquardt ICP (LM-ICP)

Instead of closed form solutions, one can also perform minimization using general NLS algorithms, as was done first in [Fit03], where the LM algorithm was used to minimize point to point (3.1) errors, termed LM-ICP. This iterative approach is slower than the closed form solution, but allows to incorporate robust cost functions and has a wider basin of convergence, due to the multiple evaluations of the cost in the inner iterations after each Levenberg step (2.28).

#### Multiview Levenberg-Marquardt ICP (MV-LM-ICP)

So far, we were always doing pairwise ICP, resulting in a relative transformation that aligns one cloud with the other. But why not align multiple clouds with each other at the same time? This is the idea of MV-LM-ICP [FCF12], the straightforward extension of LM-ICP [Fit03] from the pairwise to the multiview setting.

In the multiview setting, the roles of model and data cloud are no longer fixed, since one cloud can take on the roles of both of them. Instead of computing the relative transformation $g = (\mathrm{R}, \mathbf{t})$ that aligns the data with the model cloud, we need to estimate absolute transformations for both clouds that bring them into alignment. For example, the point to point (3.1) error between cloud $C_h$ and cloud $C_k$ with absolute poses $g_h$ and $g_k$ reads as

follows:

$$E(g_h, g_k) = \sum_{i=1}^{N_h} \left\| \mathrm{R}_h \mathbf{p}_i + \mathbf{t}_h - (\mathrm{R}_k \mathbf{p}'_i + \mathbf{t}_k) \right\|^2 \quad \text{absolute pose error} \tag{3.3}$$

where $\{\mathbf{p}_i \rightarrow \mathbf{p}'_i\}$ are the $N_h$ closest point correspondences obtained from the clouds. Let $C_1, \dots C_M$ be the set of point clouds that are brought to be in alignment. To generalize and formalize the notation of which cloud gets registered to which other cloud, we can encode these relations as a directed graph, with the adjacency matrix $A \in \{0,1\}^{M \times M}$, such that $A(h,k) = 1$ if cloud $C_h$ can be registered to cloud $C_k$. Let $g_1, \dots g_M$ be the absolute camera poses of each view in the global reference frame. The overall alignment error, which we want to minimize at this stage, is obtained by summing up the contribution of every pair of overlapping views:

$$E(g_1, ..., g_M) = \sum_{h=1}^{M} \sum_{k=1}^{M} A(h,k) \sum_{i=1}^{N_h} \left\| \mathrm{R}_h \mathbf{p}_i + \mathbf{t}_h - (\mathrm{R}_k \mathbf{p}'_i + \mathbf{t}_k) \right\|^2 \quad \text{multiview error} \tag{3.4}$$

The solutions $g_1, ..., g_M = argmin(E)$ are the absolute camera poses that align the $M$ clouds in a least squares sense. In contrast to the pairwise case, there are no closed form solutions in the multiview setting. However, rigid point cloud registration is just an instance of NLS and can thus be solved iteratively, e.g. using Ceres Solver [AM].

### Generalized ICP (GICP)

New methods to minimize correspondence errors also accelerated the development of a new error metric that doesn't have a closed form solution. The most prominent of these is GICP [SHT09], where the authors extend the point to plane (3.2) error to the symmetric plane to plane (3.6) error.

The idea is that we know the position of a point along its normal with high confidence, while we are unsure about its location in the plane. But since two non-parallel planes in $\mathbb{R}^3$ always intersect, the derivation involves the covariances $\Sigma$ of the local plane approximations to match, not just their normals. To compute surface normals and covariances, the authors used PCA. However, they did not use the covariance matrix (2.22) $\Sigma$ directly, but computed its eigen decomposition (2.23) $\Sigma = \mathrm{UDU}^T$ and replaced its eigenvalues in the diagonal matrix D to get the modified covariance matrix $\hat{\Sigma}$, imposing a disc shape prior:

$$\hat{\Sigma} = \mathrm{U} \begin{bmatrix} \epsilon & & \\ & 1 & \\ & & 1 \end{bmatrix} \mathrm{U}^T \quad \text{disc shape prior} \tag{3.5}$$

The matrix U, consisting of the orthogonal eigenvectors as columns, is left untouched, since it forms a rotation matrix which maps from the identity frame to the frame aligned with the surface normal $\mathbf{e}_0$. The smallest eigenvalue $\lambda_0$ is replaced with an even smaller constant $\epsilon = 0.001$ describing the thickness of a noise-free planar surface, while the other two larger eigenvalues are replaced with 1. This allows corresponding points to deviate further from each other in their common tangent plane than their original noisy covariance

matrices would have allowed for.

The covariance matrices of both data $\mathbf{p}$ and model point $\mathbf{p}'$ are modified in this way which finally allows to define the plane to plane error:

$$E = \sum_{i=1}^{N} d(\mathbf{p}_i, \mathbf{p}'_i)^T (\mathrm{R}\hat{\Sigma}_{\mathbf{p}_i}\mathrm{R}^T + \hat{\Sigma}_{\mathbf{p}'_i})^{-1} d(\mathbf{p}_i, \mathbf{p}'_i) \qquad (3.6)$$
plane to plane

where $d(\mathbf{p}_i, \mathbf{p}'_i) = \mathrm{R}\mathbf{p}_i + \mathbf{t} - \mathbf{p}'_i$ is the usual point to point (3.1) error vector between the points. The gist of the plane to plane (3.6) error metric is the scaling of this error vector by the information matrix $\hat{\Sigma}^{-1}_{\mathbf{p}_i \cup \mathbf{p}'_i} = (\mathrm{R}\hat{\Sigma}_{\mathbf{p}_i}\mathrm{R}^T + \hat{\Sigma}_{\mathbf{p}'_i})^{-1}$, which is the inverse of the sum of two covariance matrices. This matrix sum tries to describe the joint distribution of the two local plane approximations. Note that the data covariance $\hat{\Sigma}_{\mathbf{p}_i}$ needs to be rotated into the frame of the model covariance $\hat{\Sigma}_{\mathbf{p}'_i}$, in which the error vector $d(\mathbf{p}_i, \mathbf{p}'_i)$ is also given.

The symmetric nature of the plane to plane (3.6) error allows the algorithm to converge even in an extreme case (fig. 3.2) in which point to plane (3.2) would get stuck in a local minima.

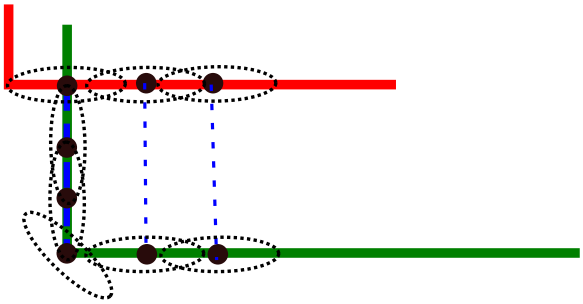


Figure 3.2: Illustration of plane to plane. Points are denoted as black dots with their covariance ellipses. Correspondences are shown as blue dashed lines. *In this case all of the points along the vertical section of the green scan are incorrectly associated with a single point in the red scan. Because the surface orientations are inconsistent, plane-to-plane will automatically discount these matches: the final summed covariance matrix of each correspondence will be isotropic and will form a very small contribution to the objective function relative to the thin and sharply defined correspondence covariance matrices. An alternative view of this behavior is as a soft constraint for each correspondence. The inconsistent matches allow the red scan- point to move along the x-axis while the green scan-points are free to move along the y-axis. The incorrect correspondences thus form very weak and uninformative constraints for the overall alignment.* [SHT09]

### Normal ICP (NICP)

In [SG15], the authors extend the point to plane (3.2) error in a different way. The coordinates of points are augmented with surface normals for both data and model cloud. These are not only used when searching for correspondences, but also in the error metric. Thus, they minimize a distance between 6D vectors instead of 3D points. This allows to solve an additional DoF in the surface orientations and has a larger basin of convergence than the original point to plane (3.2) variant.

In GICP, a $3 \times 3$ covariance matrix was obtained as the sum of the covariance matrices of both points resulting in the symmetric plane to plane (3.6) error. The inverse of the measurement covariance, the information matrix, is used to scale the error appropriately by its uncertainty. In NICP, the covariance matrix is a $6 \times 6$ matrix because of the augmented error vector. It is obtained using just the information from one of the points which makes the error unsymmetric, which is why it is an extension of the point to plane (3.2) and not of the plane to plane (3.6) error, even though the difference between both normals is minimized.

Nevertheless, the authors explain clearly an important detail which was left unexplained in GICP, namely why it is important to impose the disc shape prior (3.5). The reason is that because of sensor noise, even planar surfaces will not have zero curvature (2.24), captured by the ratio of eigenvalues. When summing up noisy covariance estimates, the resulting information matrix will become uninformative. To reduce the effect of sensor noise onto the error metric, the prior knowledge is incorporated.

## 3.2 Actuated 2D LiDAR based SLAM

The groundbreaking work of [BZ09, BZF12, KZB16] between 2009 and 2015 forms the basis of our approach. They were among the first to publish a 3D SLAM algorithm that works with an actuated 2D LiDAR on a moving platform. An initial trajectory estimate is refined in a time-windowed fashion using a specifically developed ICP variant stated as a NLS (2.26) problem. The algorithm seeks to estimate the continuous trajectory over two sweeps instead of a relative transformation between two scans. It can be summarized by the following two steps, modified from the original ICP steps (section 3.1), which are repeated until convergence:

I Compute **correspondences** between two sweeps, i.e. for each surfel $a_i$ in sweep a, transformed into world coordinates by the current trajectory estimate at the surfel's mean timestamp $T(\tau_{a_i})$, find its closest surfel in sweep b, transformed via $T(\tau_{b_i})$.

II Update the current **trajectory** estimate $T = \delta T \oplus T_{prev}$ via small corrections $\delta T$ that minimize the alignment errors between corresponding surfels.

The notion of sweep and surfel follow (section 3.2.1). Details of this algorithm and its improved version (section 3.2.2) are discussed in the light of our ICP taxonomy (section 3.1.1). The author's latest journal article (section 3.2.3) employs the same algorithm on a quadrocopter. The method presented here introduces some useful concepts, but nevertheless has several flaws, which will be discussed in the main part of this work (section 4.1).

### 3.2.1 Spinning Laser 2009

In the first conference paper [BZ09] the authors mount the 2D laser scanner SICK LMS-291 (fig. 2.2a) on a spinning platform (fig. 1.1b). The device is rotated around the center of its scan plane at a rate of $\frac{1}{2}$Hz. The data collected in one half-revolution (sweep), lasting 1s, covers the entire space visible to the sensor. An accurate encoder on the spinning platform gives the orientation of the LiDAR scanner with respect to the platform, which allows initially aggregating the scans from one sweep. However, significant platform motion can occur during the duration of a sweep, so the platform trajectory during this time

must also be taken into account. Instead of using a traditional scan-matching algorithm which registers rigid 2D LiDAR scans, the authors thus developed a sweep-matching algorithm registering deformable 3D sweeps. The sweep-matching algorithm is an extension of the ICP scan-matching algorithm applied to **surf**ace **el**ement (surfel)s. These are local surface features with strong shape characteristics (section 4.5.2), either planar or cylindrical, computed for each sweep. Modifications to standard ICP, according to the taxonomy (section 3.1.1) are the following:

1. **Selection**: Instead of the original laser scan points, surfels are used. These are generated as follows: For each sweep, the space is discretized into a pyramid of 3D voxelgrids ranging from $0.5m$ to $8m$. Each pyramid level additionally consists of voxels offset by half the voxel size to account for quantization artifacts. For all the points within a sweep that fall in the same voxel, their centroid as well as their covariance matrix is calculated. If the covariance matrix suggests that the voxel describes planar or cylindrical features and if there are a sufficient number of points inside the voxel (for very fast motion undersampling occurs), a surfel is generated.

2. **Matching**: Corresponding surfels between two sweeps are matched based on the Euclidean distance between 9D vectors formed by concatenating centroid $\boldsymbol{\mu}$, normal $\mathbf{e}_0$ and cylinder axis $\mathbf{e}_2$, with appropriates weights of each 3D component vs the other, of a surfel.

3. **Weighting**: Matches are weighted by $\sqrt{\frac{\lambda_2}{\lambda_0}}(1 - \frac{\lambda_0}{\lambda_2})$ to account for uncertainty in the normal direction. To account for outliers, Cauchy weights are applied as robust Lorentzian cost function to the residuals in an M-estimator framework.

4. **Rejecting**: nothing is rejected

5. **Error metric**: First, one has to compute the eigen decomposition (2.23) of the matched surfels covariance matrices $\Sigma_{a_i \cup b_i} = \Sigma_{a_i} + \Sigma_{b_i}$ from sweeps a and b.
   - Planar surfels should have:
     a) their centroids aligned $\vec{\mathbf{n}}^T_{a_i \cup b_i}(\boldsymbol{\mu}_{a_i} - \boldsymbol{\mu}_{b_i}) = 0$ in direction of their common surface normal $\vec{\mathbf{n}}_{a_i \cup b_i} = \mathbf{e}_0/\sqrt{\lambda_0}$. [1]
     b) their normal directions aligned $\vec{\mathbf{n}}_{a_i} \times \vec{\mathbf{n}}_{b_i} = \mathbf{0}$
   - Cylindrical surfels should have their centroids aligned $C^T_{a_i \cup b_i}(\boldsymbol{\mu}_{a_i} - \boldsymbol{\mu}_{b_i}) = 0$ orthogonal to the cylinder axis: $C_{a_i \cup b_i} = [\mathbf{e}_0/\sqrt{\lambda_0}, \mathbf{e}_1/\sqrt{\lambda_1}]$

   Scaling the eigenvectors by the inverse square root of their corresponding eigenvalues aims to consider the measurement uncertainty.

6. **Minimization**: Minimization is carried out by first discretizing the trajectory corrections over a sweep interval and linearly interpolating in between sample poses, and linearizing the constraints according to the first-order Taylor expansion of an infinitesimal rotation (2.2): $\delta\mathrm{R} = \mathrm{I} + [\delta\mathbf{r}]_\times$. Then, the problem is solved iteratively via a weighted NLS (2.26) optimization. Finally, the corrections are applied to the

[1] Note the similarity to the point to plane (3.2) error metric, except that the common normal is used instead of the normal of just one surfel.

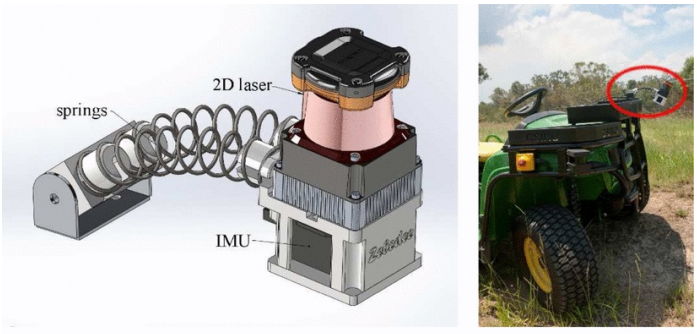
previous trajectory estimate and then a cubic spline is used to reconstitute a smooth, continuous trajectory.

**Additional constraints:** The sweep-matching algorithm differs from standard ICP, because additionally to the surfel matches, further constraints are added to the optimization problem:

- Smoothness constraints: $(T_{i+1})^{-1} \oplus T_i = (T_i)^{-1} \oplus T_{i-1}$
- Initial conditions: the velocity at the beginning of sweep b is constrained to maintain continuity with the velocity at the end of sweep a.

**Initialization:** The trajectory of sweep a is initialized with that of the previous sweep, the trajectory of sweep b with a motion model that decelerates from the initial velocity at the end of sweep a to zero velocity.

### 3.2.2 Zebedee 2012




(a) CAD model (b) Vehicle mounted

Figure 3.3: Zebedee - a spring-mounted 3D range-sensing system [BZF12].

In their second conference paper [BZF12] the authors removed the spinning platform and instead mount the LiDAR scanner together with an IMU on a spring, which is passively actuated by the movement of a vehicle or even a walking person. The resulting irregular and nondeterministic motion of the sensor head provides some challenges, in particular, there is no notion of a sweep anymore. Instead of the encoder, the IMU is now used to initially aggregate the scans and initialize the trajectory. The differences to the sweep-matching algorithm (section 3.2.1) of their previous work [BZ09], according again to the taxonomy (section 3.1.1) are the following:

1. **Selection**: Since there are no more sweep, space and time are discretized into a 4D voxel-grid, where each 3D spacial cell can contain multiple clusters of points, depending on their discretized time values. Thus, the allowed timespan of points inside a cluster can be seen as a *nominal sweep period*, because the spring will still induce some form of periodic movement, e.g. nodding where the elevation angle approximately forms of a sine-wave. Only planar surfels are used, not cylindrical ones. Clusters with a small rotational velocity component normal to the scan plane are discarded.

2. **Matching**: Corresponding surfels are matched based on the Eclidean distance between 6D vectors formed by concatenating centroid and normal $\mathbf{e}_0$. This is also the first time they mention approximate nearest neighbor queries in a kd-tree to get these matches. Since there are no more sweeps, we can't match sweep a against sweep b, but instead correspondences can arise from any two surfels if their timestamps differ by more than half of the *nominal sweep period*. These quality matches over a larger time span provide more useful constraints for trajectory optimization.

3. **Weighting**: The soft outlier threshold $\bar{r}$, used to compute the weights $w_i$ via the Lorentzian function $w_i = (1 + (\mathbf{r}_{ij}/\bar{r})^2)^{-1}$ on the residual $\mathbf{r}_{ij}$, is initially large and decreased at each iteration to increase the basin of convergence.

4. **Rejecting**: only reciprocal surfel matches are retained

5. **Error metric**: Instead of three error metrics, only one is used. Planar surfels should have their centroids aligned $\vec{\mathbf{n}}_{a_i \cup b_i}^T (\boldsymbol{\mu}_{a_i} - \boldsymbol{\mu}_{b_i}) = 0$ in direction of their common surface normal $\vec{\mathbf{n}}_{a_i \cup b_i} = \mathbf{e}_0 / \sqrt{\sigma_r + \lambda_0}$, where the sensor measurement noise $\sigma_r$ is further incorporated into the error metric compared to the previous paper.

**Additional constraints:** The trajectory is now processed in a time-windowed segment, which is advanced by a fraction of its length as new data arrives. This sweep-matching algorithm differs from the previous one [BZ09] by incorporating the following additional constraints into the optimization problem:

- Initial conditions: instead of constraining the velocity at the border of consecutive sweeps, continuity is enforced by penalizing any changes to the first three trajectory correction samples in the current time-windowed segment.

- IMU measurement deviations are used instead of smoothness constraints. The errors on translational acceleration and rotational velocities are as follows:

$$ {}_s\mathbf{e}_{\boldsymbol{\omega}}(\tau) = \Sigma_{\boldsymbol{\omega}}^{-1/2} \left( {}_s\tilde{\boldsymbol{\omega}}(\tau) - \frac{d\mathbf{r}(\tau)}{d\tau} \right) \tag{3.7} $$

$$ {}_w\mathbf{e}_{\mathbf{a}}(\tau) = \Sigma_{\mathbf{a}}^{-1/2} \left( \mathbf{r}(\tau) \oplus {}_s\tilde{\mathbf{a}}(\tau) - \frac{d^2 {}_w\mathbf{t}(\tau)}{d\tau^2} - {}_w\mathbf{g} \right) \tag{3.8} $$

  Note that ${}_w\mathbf{e}_a(\tau)$ is expressed in the world frame ${}_w$, but ${}_s\mathbf{e}_{\boldsymbol{\omega}}(\tau)$ in the IMU sensor frame ${}_s$. The second derivative of the trajectory ${}_w\ddot{\mathbf{t}}(\tau) = \frac{d^2 {}_w\mathbf{t}(\tau)}{d\tau^2}$ with respect to time $\tau$, as well as the gravity vector ${}_w\mathbf{g}$ are both expressed in the world frame ${}_w$, thus we have to rotate the raw acceleration measurement ${}_s\tilde{\mathbf{a}}$ into the world frame ${}_w$ before subtracting these quantities. $\Sigma$ denotes the measurement covariance, from which we take the inverse square root so that the squared norm $e^T e$ of the residual $e$ is scaled by the information matrix $\Sigma^{-1}$.

- IMU bias estimation: small correction updates to the 6 DoF bias vector on the raw acceleration and rotational rate measurements are included in the state and considered when computing the IMU measurement deviations.

- Timing latency estimation: Since they are now using two sensors instead of one, unsynchronized clocks in both devices will have a severe effect on the outcome. Thus the optimization state is further augmented with an extra dimension to estimate timing errors between IMU and LiDAR.

**Initialization:** The first trajectory segment is initialized by integrating accelerometer and gyro measurements from the IMU. During the time-windowed optimization, IMU data is only required to propagate the trajectory for the remainder of the current time window.

**Operating modes:** To reduce the drift during the time-windowed processing, the authors suggest to include surfel matches over longer timespans, what they call fixed views. In addition to processing the trajectory sequentially in time windows, one can also optimize the entire trajectory at once in a subsequent global optimization step, given a good initial guess. This should improve the global consistency of the reconstructed trajectory.

### 3.2.3 Bentwing 2016

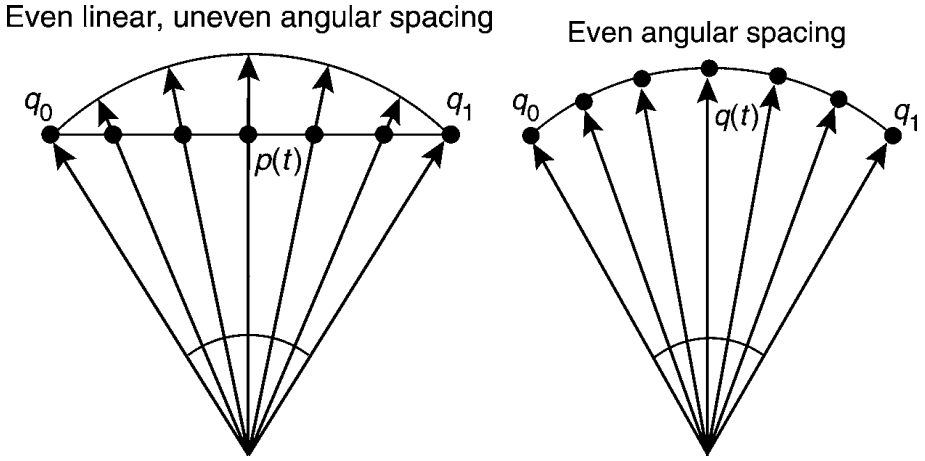
In the latest journal article [KZB16] the laser scanner is mounted below a quadrocopter and passively actuated by the downdraft from the blades to perform a rotational motion similar to the first work [BZ09]. However, the authors employed the algorithm of the previous work [BZF12] with only minor modifications by **rejecting** (4.) grossly distant matches, which is actually a crucial step in ICP to make it converge properly. It allows to get rid of wrong correspondences after the transformation update, which can only partially be corrected for with robust cost functions. **Initialization** of the IMU bias is done while the platform is stationary before takeoff. **Additional constraints** are added to the optimization problem to estimate the physical offset between LiDAR and IMU base frames to account for any inaccuracies during calibration.

## 3.3 Interpolating orientations

The need to interpolate smoothly between orientations comes originally from the computer graphics domain, in particular for keyframe or character animation. It was a major research topic from 1985 until 1995. However it took about 20 years for these ideas to be adopted in the computer vision domain, which usually has the inverse problems to solve. In this section, we will thus revisit the classical research done about this topic in the computer graphics domain so that we can apply it to our approach later on. The background for this section is a good understanding of quaternions when used to represent orientations (section 2.1.2) and of the usual interpolation methods in Euclidean space (section 2.3). In this section we will present four different orientation interpolation methods called renormalized LERP, SLERP, SQUAD and a cumulative cubic B-spline approach. These will be evaluated later on (section 4.2.3) to help us pick a suitable trajectory representation needed for our approach.

### Renormalized LERP

**L**inear int**erp**olation (LERP) can be used to interpolate between two orientations, given as the two unit quaternions $\mathbf{q}_0, \mathbf{q}_1$. We just need to apply LERP (2.16) to the 4 quaternion components. But then, the intermediate quaternions $\mathbf{q}(u)$ would not be of unit length anymore. Thus we have to renormalize the quaternion by dividing it through $\|\mathbf{q}(u)\|$. Unfortunately, the resulting unit quaternion curve will not appear smooth, because the intermediate quaternions are unequally spaced, meaning the angular velocity along the path varies (fig. 3.4a).




(a) renormalized LERP: The straight line connecting the two quaternions is divided into line segments of equal length. Projection lines from the center of the circle through the segment boundaries are drawn. The angle between the projection lines is unequal and so are the lengths of the circular arc segments.

(b) SLERP: The angle between the two quaternions is divided into equal parts. The lengths of the circular arc segments are thus also equal.

Figure 3.4: Two different methods to interpolate between 2 unit quaternions $\mathbf{q}_0, \mathbf{q}_1$. The space of unit quaternions is spherical due to the unit constraint, which is why circular arcs are shown here. [Han06, Fig 10.4, 10.5]

### Spherical linear quaternion interpolation (SLERP)

The correct way to interpolate two unit quaternions is to do the interpolation with $u \in [0, 1]$ on the surface of the sphere $\mathbb{S}^3$, which [Sho85] termed SLERP:

$$slerp(\mathbf{q}_0, \mathbf{q}_1, u) \quad = \quad \mathbf{q}_0(\bar{\mathbf{q}}_0\mathbf{q}_1)^u \quad = \quad \mathbf{q}_0 \exp(u \log(\bar{\mathbf{q}}_0\mathbf{q}_1)) \tag{3.9}$$
SLERP

Application of SLERP (3.9) produces a geodesic in $\mathbb{S}^3$, with the property of constant angular velocity along the corresponding path in $\mathrm{SO}(3)$. This is due to the fact that the intermediate quaternions are equally spaced on this geodesic, which is a path on the surface of a sphere (fig. 3.4b). If interpolating more than 2 orientations, a piecewise application is possibly, but provides only $C^0$ continuity, just as its counterpart in Euclidean space.

### Spherical cubic spline quadrangle quaternion interpolation (SQUAD)

To gain higher order continuity, the idea of uniform composite cubic bezier curves is applied to quaternion interpolation in [Sho87]. Similar to the de Casteljau construction, but using SLERP (3.9) instead of LERP (2.16), a cubic Bezier curve segment is defined by the 4 control quaternions $\mathbf{q}_i, \mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{q}_{i+1}$. To ensure $C^1$ continuity across consecutive segments, the inner control quaternions $\mathbf{s}_i, \mathbf{s}_{i+1}$ are computed in a special way from 3 given quaternions of the original sequence to interpolate. For $\mathbf{s}_i$, the idea is to approximate the derivative or angular velocity at $\mathbf{q}_i$ by central differences, using $\mathbf{q}_{i-1}, \mathbf{q}_{i+1}$, and then placing $\mathbf{s}_i$ to assure $C^1$ continuity in quaternions space. The same is done for $\mathbf{s}_{i+1}$ at $\mathbf{q}_{i+1}$ Since the original paper [Sho87] is not available anymore, the definition of SQUAD, using quaternion exponentiation (2.5) and logarithm (2.6), is replicated in [DKL98, Def. 17]:

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

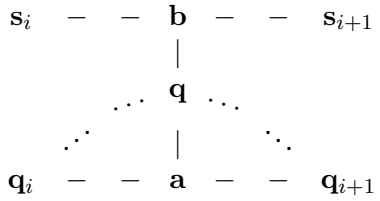
$$squad(\mathbf{q}_i, \mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{q}_{i+1}, u) = slerp\big(slerp(\mathbf{q}_i, \mathbf{q}_{i+1}, u), slerp(\mathbf{s}_i, \mathbf{s}_{i+1}, u), 2u(1-u)\big) \quad (3.10)$$

SQUAD

Note that the iterated application of slerp in squad is a little different than the construction of a cubic Bézier curve (2.17) using the de Casteljau construction, which can be expressed using iterated slerp as follows:

$$bezier(\mathbf{q}_i, \mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{q}_{i+1}, u) = slerp\big(slerp(\mathbf{q}_i, \mathbf{s}_i, u), slerp(\mathbf{s}_{i+1}, \mathbf{q}_{i+1}, u), u\big)$$

In contrast to that, the construction of squad is actually a *parabolic blending* using bilinear interpolation, which is nicely explained in [WW91]: Consider the control polygon, a rectangle made up by the control vertices $\mathbf{q}_i, \mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{q}_{i+1}$ in the u-v coordinate system with $\mathbf{q}_i$ at the origin:




The two inner slerps of squad interpolate linearly with an amount of $u \in [0, 1]$ along the horizontal edges to get the intermediate points $\mathbf{a}, \mathbf{b}$. These will then be interpolated vertically by the outer slerp with an amount of $v = 2u(1-u)$ to get the final point $\mathbf{q}$. Note that $v$ is quadratic in $u$, it is 0 for $u = 0$ and $u = 1$, thus the final curve starts at $\mathbf{q}_i$, ends at $\mathbf{q}_{i+1}$, has its maximum height of $v = 0.5$ at $u = 0.5$ and thus traces out a parabola, as indicated by the dots.

### Cumulative B-spline quaternion curve

The curve given by SQUAD (3.10) is $C^1$ continuous and has local control. Nevertheless, a change in a control point requires to recompute some interior control points to maintain this continuity. This becomes tedious when optimizing over trajectories. Ten years later, the idea of Shoemake is generalized to arbitrary basis functions, gaining $C^2$ continuity for

free, with the introduction of the transformation to cumulative basis form [KKS95]. Since this is the curve we will be using, we will discuss it later (section 4.2.2) in depth.

## 3.4 Continuous time estimation

In this section, the theory and recent progress in continuous time estimation is presented. We start with an article motivating the need for such methods, and then discuss four different publications that apply the theory in different applications. Special focus is given to the used rotation parameterization and interpolation method. Some of the findings of these works are incorporated into our approach later on.

#### Continuous time SLAM (CT-SLAM)

In the conference paper [FBS12], the authors motivate the need for continuous time estimation methods in robotics because:

- High-rate sensors such as IMU's capture data at high rate, which in discrete estimation methods would require to include a pose variable in the state for each measurement, making it very large.
- Continuously capturing devices such as sweeping LiDAR scanners or rolling shutter cameras, when moved during acquisition, produce distortion artefacts if their measurements are handled as discrete snapshots in time.

They bridge the gap between SLAM and continuous-time estimation theory, which has existed for longer than SLAM, by providing a derivation of SLAM in continuous time, which they call CT-SLAM. As an specific implementation, they use a B-spline basis representation of the trajectory with the favourable property of local support and simple analytic derivates, needed to synthesize IMU measurements. The rotation is represented minimally using the Cayley-Gibbs-Rodrigues vector, a minimal global representation of rotations that has - like all minimal representations of rotations - singularities. For evaluation purposes, the authors pursue an IMU to camera calibration experiment.

In their subsequent journal article [FTBS15], they additionally give a detailed overview over current continuous-time state estimation methods and add another experiment where the IMU data is used to compensate rolling shutter artefacts of a moving camera and compare the results to a global shutter camera which is mounted next to it.

#### Continuous ICP (CICP)

The authors of [ABB14] use the same interpolation method and rotation parameterization, with the only minor difference of implementing the B-spline basis functions using the de Boor algorithm instead of the matrix definition.

They apply it to registering scans using ICP from actuated LiDAR, which they term CICP. This is similar to the work of [BZ09] as discussed in-depth in the previous section, with the following differences:

1. Due to the global parameterization of rotations, the authors are no longer limited to estimating small motion corrections, a constraint arising from the linear small-angle approximation (2.2) used to represent orientations in [BZ09].

2. Using a B-spline instead of linear interpolation automatically ensures smoothness across segment boundaries, which [BZ09] had to enforce in the optimization through additional smoothness constraints.

3. However, [ABB14] minimizes point-to-point correspondences instead of surfel matches, which might be faster to compute, but narrow the basin of convergence.

Their experimental findings suggest that the B-spline basis has to be at least of degree three, i.e. a cubic spline. Furthermore, when outliers in the point-to-point matches are present, regularization via robust L1 cost functions improves accuracy. In their experiment with a spinning Hokuyo laser scanner (fig. 2.2c), they use Visual Odometry (VO) instead of an IMU to initially undistort the scans.

#### Spline Fusion

In [LPPS13, PPLS15], the authors employ continuous time estimation for rolling shutter camera calibration and visual-inertial SLAM. As a continuous-time representation, they use the cumulative cubic B-spline approach of [KKS95], but applied to SE(3) and its Lie algebra se(3) instead of quaternions. This allows for straightforward visual-inertial fusion, because a generative model for synthetic inertial measurements using the analytic derivatives of this trajectory representation is also derived. It allows to incoorporate IMU measurements naturally into the estimation process, which act as a regularizer for the visual feature matches. The authors choose the SE(3) parameterization because it is free from singularities and models torque-minimal trajectories. Originally, we wanted to implement this trajectory representation for use in our own work. They provide the full derivation and detailed formulas needed for the trajectory derivatives needed to synthesize IMU measurements, which helps a lot in the implementation process. However, we found out that these torque minimal trajectories, as implemented in the paper, have the unconvenient property of coupling orientation with translation (section 4.2.4). Also, there is a lot of matrix multiplication involved to compute the derivatives. Due to the redundency of this parameterization, this slows down optimization when residuals depend on time derivatives of the trajectory. Even worse, it is not described how convert the result of the matrix multiplication back to an angular velocity vector.

#### Relative continuous-time SLAM

This line of work [AB13, AMB15] in particular addresses the performance issues of incoorporating loop-closures when an absolute trajectory representation is used. When a loop closure occurs, all of the trajectory base functions have to be re-evaluated to propagate this change. In discrete-time, the problem was overcome by introducing a relative pose-graph, in which loop-closures take only constant time to incorporate because only a constant size subset of all the relative poses have to be updated. To convert this idea to continuous time, the authors look at the infinitesimal of relative poses, linear and angular velocities.

They propose to use a continuous time 6D velocity curve, concatenating linear and angular velocities and interpolating with a standard cubic B-spline. This is a truly minimal representation of rigid body motion and avoids singularities. However, the benefit of constant time updates in the case of loop-closures comes with the cost of having to integrate the whole trajectory up to a specific point in time when an absolute pose is needed.

#### Attitude estimation using B-spline on Lie Groups

The journal article [SFSF15] extends the theory of cumulative cubic B-spline unit quaternion curves 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 expressivenes of the curve. In particular, by increasing the spline order from 4 to 5, one can half the number of segments (from 20 to 10), or equivalently double the knot spacing, and still get the same accuracy. To my knowledge, all previous continuous time estimation methods silently concluded that $C^2$ continuity of the trajectory is enough to accuartely model the acceleration of IMU's. However, a $C^2$ continuous trajectory is only piecewise linear when derived twice, and can thus only model linear interpolation of accelerations between two knots. A higher order continuous curve allows for more freedom of the acceleration curve between two knots and thus enables to increase the knot spacing. It is thus expected that future research pays more attention to the 5th or 6th order B-spline.

Furthermore, the authors have the nice idea to model the slowly time-varying IMU bias error with a 3D B-spline, sharing the same knots as the continuous trajectory. To regularize this spline so that it is really slowly varying, they add the derivative of this spline as an error term into the cost function to optimize, pulling the derivatives towards zero.

# 4 Approach

After having covered the theoretical background and previous work, we finally come to the development of our main algorithm. It achieves improvement compared to existing approaches by carefully combining different ideas (section 4.1). In (section 4.2), we evaluate several continuous time trajectory representations in an experimental manner before choosing the one that suits us best. The high-level overview of our algorithm as well as the pseudo-code is given (section 4.3), before diving into the details: We show how IMU measurements can be synthesized from our trajectory representation (section 4.4) and how they can be used to initialize the trajectory. The next section shows how surfel matches originating from LiDAR can be incorporated (section 4.5). The overall model for our NLS problem and its optimization (section 4.6) concludes this chapter.

## 4.1 Motivation

The main inspiration [BZ09, BZF12, KZB16] for our algorithm was previously presented (section 3.2). In the following, we address certain limitations and suggest alternatives that are beneficial for our final algorithm.

**Use of infinitesimal rotations.** Due to singularities, the authors are limited to estimating trajectory corrections instead of the full trajectory. In each publication, a slightly different trajectory representation and update rule is employed:

$$
\begin{aligned}
[\text{BZ09}] : T &= [\delta\mathbf{r} \oplus \mathbf{r} \ , \delta\mathbf{t} + \delta\mathbf{r} \oplus \mathbf{t}]^T && \equiv \delta T \oplus T_{prev} \\
[\text{BZF12}] : T &= [\delta\mathbf{r} \oplus \mathbf{r} \ , \delta\mathbf{t} + \mathbf{t}]^T && \equiv T^{\mathbf{t}}_{prev} \oplus \delta T \oplus T^{\mathbf{r}}_{prev}, T_{prev} = \ T^{\mathbf{t}}_{prev} \oplus T^{\mathbf{r}}_{prev} \\
[\text{KZB16}] : T &= [\delta\mathbf{r} \oplus \mathbf{r}_0, \delta\mathbf{t} + \mathbf{t}_0]^T && \equiv T^{\mathbf{t}}_{base} \oplus \delta T \oplus T^{\mathbf{r}}_{base}, T_{base} = [\mathbf{r}_0, \mathbf{t}_0]
\end{aligned}
$$

Because of the infinitesimal rotation (2.2) approximation errors, the authors furthermore had to change the rigid body motion composition: In particular, the update on the translation part $\delta\mathbf{t}+\delta\mathbf{r}\oplus\mathbf{t}$ [BZ09] is done without rotating first $\delta\mathbf{t}+\mathbf{t}$ in [BZF12]. The authors claim that this non-standard formulation of rigid body motion composition is preferable, since translations are expressed in the global frame instead of the sensor frame. The term $\delta\mathbf{r} \oplus \mathbf{t}$ is problematic for large $\mathbf{t}$ due to errors introduced via the approximation. In the last paper [KZB16], the trajectory is decomposed into a baseline trajectory and small corrections, which allows only small corrections to be modeled correctly because of the linearization involved.

These limitations can be resolved with **unit quaternions**, which are singularity free and can be employed in optimization using their related Lie algebra of pure quaternions as local parameterization.

**Use of linear interpolation** to compute the Jacobians and of a spline to evaluate the trajectory since $C^2$ continuity for the IMU measurements is required. This actually means using derivatives of a different function in the optimization than for its evaluation. Furthermore, the usage of the spline is only mentioned briefly leaving important questions regarding data fitting and implementation details open.

*To discretize the trajectory over a sweep interval, we compute poses at increments of 0.2s and later use a cubic spline to reconstitute a smooth, continuous trajectory. [...] Due to the discretization of the trajectories, and the fact that the voxel timestamps are unlikely to fall exactly on the trajectory sample times, the match constraints are linearly interpolated between the closest two trajectory samples.* [BZ09]

*For purposes of computing the Jacobian, we assume that the trajectory corrections are linearly interpolated in time between the trajectory samples.* [BZF12]

*Implementationally, trajectories (and corrections) are stored as samples at a reasonable frequency for the motion bandwidth (e.g. 100 Hz) and a spline interpolates transformations for times between the samples. [...] One key difference between our solution and the others is that ours considers corrections to the trajectory in the state rather than the trajectory itself. Because the corrections can be adequately represented at a lower frequency, this allows us to reduce the number of variables (spline knots) required in the state while still maintaining high-frequency information from the trajectory.* [KZB16]

Instead, we propose to use a **cumulative B-spline** for interpolation, both to compute the Jacobians and to evaluate the trajectory. Maintaining high-frequency information from the IMU through the use of a static baseline trajectory with a tight knot spacing and a lower frequency trajectory made up of corrections that serve as the state seems like a good idea. However, to initialize the static high frequency trajectory from the IMU measurements, one usually has to solve an optimization problem with a large state anyways. Using one medium frequency trajectory and giving it additional expressiveness by using a higher order spline as suggested by [SFSF15] seems to be a good trade-off.

**Use of an inconvenient error metric** for registration which requires to compute common surface normals for each match. One needs to recompute the eigen decomposition (2.23) of a sum of covariance matrices during the ICP optimization, which is quite expensive. Furthermore, attempts to fix noise issues are not well explained.

Instead, we propose to use the plane to plane (3.6) **error metric of GICP**, where the noise issues are handled by the well-received disc shape prior (3.5). This has the additional benefit that one does not need the eigen decomposition of the sum of covariance matrices, but just the matrix inverse to get the information matrix, which is much cheaper to compute.

## 4.2 Trajectory representation

The choice of trajectory representation is an important decision affecting design but also final achievable accuracy of a 3D trajectory estimation approach such as ours in a major way. The trajectory representation is uniquely determined by the rigid body motion parameterization of the base poses (section 2.1) and the interpolation method (section 2.3).

It should fulfill some important properties to be useful for continuous time trajectory estimation (section 3.4):

- *Local control* so that an update of the trajectory estimate at a specific time only affects its surroundings.
- $C^2$ *continuity of both position and orientation* so that physical smoothness constraints can be handled naturally.
- *Orientation parameterization without singularities* to be able to represent all possible motions.
- *As few parameters as possible* to allow for efficient computation.
- *Analytic derivatives with efficient formulas* to be able to synthesize angular velocity and linear acceleration measurements and use them for fast and accurate optimization.

However, the choice of which representation to use is not easy because there is no perfect solution. Concerning interpolation methods, there is always a trade-off between (1) interpolation vs approximation, (2) high continuity, (3) local vs global control and (4) computational complexity (section 2.3). Because an orientation has three DoF, the absolute minimum number of parameters to represent it is three. However, there are no minimal parameterizations without singularities. Every extra parameter used in a singularity-free representation incurs additional constrains on the parameters that need to be maintained during interpolation and optimization (section 2.1).

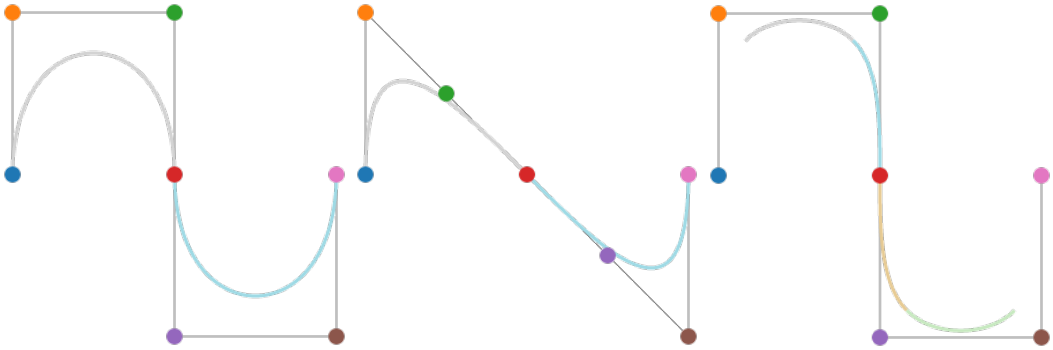
We will first discuss the confronting requirements of local control, continuity and interpolation vs approximation of the two most popular higher order interpolation methods (section 4.2.1). Then, we will give implementation details on a $C^2$ continuous approximating orientation curve without singularities and with local control (section 4.2.2). We subsequently compare it to other orientation interpolation methods (section 4.2.3). Finally, we compare two alternative approaches to arrive at our chosen trajectory representation for the full rigid body motion, consisting of orientation and translation (section 4.2.4).

### 4.2.1 Choice between B-spline and composite Bézier curve

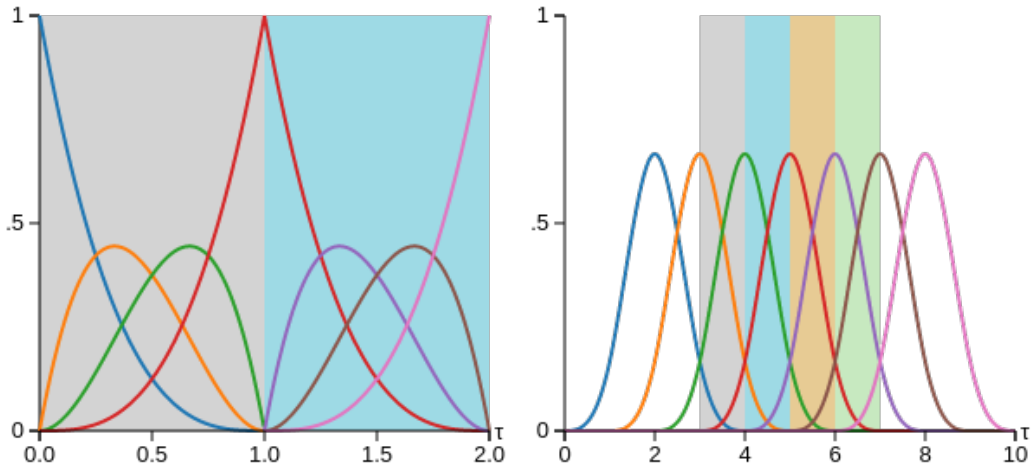
When interpolating many control points with $C^2$ continuity requirements, we can either use a B-spline (section 2.3.3) or a **composite Bézier curve**, which is a series of Bézier curves or segments, where the last control point of one segment is the first control point of the next. This ensures $C^0$ continuity. Depending on the application, additional smoothness requirements may be needed. Note that in contrast to B-spline basis functions (fig. 4.1e), the Bézier basis functions (fig. 4.1d) of one segment have no influence into neighboring segments. Thus we cannot get this continuity for free, but have to place interior control points in a special way.

$C^1$ continuous curves additionally need to have identical tangents at the segment boundaries. This can be ensured by coupling the positions of the directly adjacent inner control points on either side of the segment boundary. These three points need to form a line, with the control point joining the two segments in the middle. For example (fig. 4.1a) the green, red and violet control point form a line and the red point is in the middle.

$C^2$ continuous curves additionally need identical curvature at the segment boundaries. This requires two more interior control points than in the $C^1$ case to be placed in a special way. The vector joining these two control points has to be twice the vector joining the inner control points from the $C^1$ requirement. For example (fig. 4.1b), the vector joining the orange and brown point is twice the vector joining the green and the violet point. Note that it is not a requirement that the segment boundary control point is in its middle as it is in the example.



(a) composite $C^1$ Bezier curve (b) composite $C^2$ Bezier curve (c) B-spline




(d) composite Bezier curve basis (e) B-spline basis

Figure 4.1: Comparison of composite cubic Bezier 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.

### Local control vs interpolation

$C^2$ continuity requires at least cubic curves. However, to gain $C^2$ continuity, a cubic composite Bézier curve loses local control, because to enforce $C^2$ continuity, all the control points become dependent on each other. The constraints on interior control point placement propagate through the whole curve, so if a single control point moves, the whole

curve needs to be re-evaluated.[1]

On the other hand, cubic B-splines have $C^2$ continuity and local control, but they lose the interpolation property of a composite cubic Bézier curve, they only approximate the control points. The intuitive requirement that an interpolation method should actually interpolate the given sequence of control points is actually not a requirement in continuous time estimation, as stated in the introduction of this section. It is merely sufficient that we can control the shape of the resulting curve roughly by moving control points, which do not lie on the curve but are only approximated by the curve. This complicates initialization of the curve, because one has to initialize the control points with meaningful values. However, only locally controllable curves make trajectory estimation feasible. Only a small subset of control points has to be updated when new measurements arrive and the curve automatically stays $C^2$ continuous. Since we will be using the base poses of the trajectory representation as the state to optimize in NLS optimization, a rough initialization is usually sufficient. Constraints specified on the curve or its derivatives let it then converge iteratively to a curve that fits specified properties. So to summarize this section:

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

### 4.2.2 Cumulative B-spline quaternion curve

A B-spline (2.19) in its standard form is given as sums of basis functions with control points as coefficients (2.19): $\mathbf{p}(\tau) = \sum_{i=0}^{n} N_i^k(\tau)\mathbf{p}_i$ . Using the cumulative basis $\tilde{N}_i^k(\tau) = \sum_{j=i}^{n} N_j^k(\tau)$ with the property of $\tilde{N}_0^k(\tau) = 1, \tau > \tau_0$ due to the partition of unity, the B-spline can be rearranged into cumulative form as follows [KKS95]:

$$\mathbf{p}(\tau) = \mathbf{p}_0 + \sum_{i=1}^{n} \tilde{N}_i^k(\tau)(\mathbf{p}_i - \mathbf{p}_{i-1}) \tag{4.1}$$

cumulative B-spline

Instead of summing over all control points, we take the first control point as an absolute anchor point and sum up differences of the other control points, with the weights adapted to get the same result. In Euclidean space, the difference between consecutive control points $\mathbf{p}_{i-i}$ and $\mathbf{p}_i$ is the time difference between those points $\Delta\tau = \tau_i - \tau_{i-1}$ times the velocity $\mathbf{v}_i\Delta\tau = -\mathbf{p}_{i-1} + \mathbf{p}_i$.

While we cannot apply the B-spline in its standard form to the Lie group of unit quaternions (because the group is not closed under addition, meaning the weighted sum of unit quaternions might not be unit anymore), the concept of velocity also exists in quaternion space, more specifically in its associated Lie algebra of pure quaternions. These in term can be interpreted as angular velocity vectors: $\boldsymbol{\omega}_i\Delta\tau = \log(\mathbf{q}_{i-1}^{-1}\mathbf{q}_i)$.

[1] There are actually only two remaining DoF one can choose after specifying all the segment boundary control points, namely either the tangent at the first and the last control point or alternatively, the curvature. In certain applications, this can be a desired behavior. Given the boundary conditions and the segment boundary control points to interpolate, the positions of all the inner control points don't have to be supplied but can be calculated by solving an equation system incorporating the above constraints.

The Lie algebra of pure quaternions allows for multiplication by a scalar weight as needed for interpolation. However, it can only represent rotation differences instead of absolute orientations, namely a rotation around a fixed axis with constant angular velocity. Thus we need to use the cumulative form of the B-spline, applying these relative rotation differences in sequence to the first control quaternion.

Replacing $\mathbf{p}(\tau)$ with $\mathbf{q}(\tau)$, $\mathbf{p}_0$ with $\mathbf{q}_0$, $\mathbf{v}_i$ with $\boldsymbol{\omega}_i$ and 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:

$$\mathbf{q}(\tau) = \mathbf{q}_0 \prod_{i=1}^{n} \exp\left(\tilde{N}_i^k(\tau) \log(\mathbf{q}_{i-1}^{-1}\mathbf{q}_i)\right) \tag{4.2}$$

cumulative quaternion B-spline

**Uniform cubic cumulative quaternion B-spline**

When using uniform time intervals $\Delta\tau = \tau_{i+1} - \tau_i \forall i \in [0, n-1]$ as control point spacing, a uniform cubic cumulative quaternion B-spline basis can be obtained via matrix multiplication [PPLS15] :

$$\widetilde{\mathbf{N}}(u) = [u^3, u^2, u, 1]\,\mathbf{C} \quad , \quad \mathbf{C} = \frac{1}{6}\begin{bmatrix} 0 & 1 & -2 & 1 \\ 0 & -3 & 3 & 0 \\ 0 & 3 & 3 & 0 \\ 6 & 5 & 1 & 0 \end{bmatrix} \tag{4.3}$$

cumulative B-spline matrix

Note that a column in this cumulative basis form matrix is obtained by summing up the columns of the standard basis form matrix (2.20) with equal or greater index. In particular, note that the rightmost column is the same for both bases.

We want to evaluate the cubic quaternion curve $\mathbf{q}(\tau)$ at a specific time $\tau \in [\tau_i, \tau_{i+1})$. Thus we have to transform the global time variable $\tau$ to the segment local parametric variable $u$. Furthermore, we need to find out the index $i$ of the knot shortest before the current evaluation time:

$$i = \frac{\tau - \tau_0}{\Delta\tau} \quad , \quad u = \frac{\tau - \tau_i}{\Delta\tau}$$

The knot index gives us the corresponding 4 control quaternions $\mathbf{q}_{i-1}, \mathbf{q}_i, \mathbf{q}_{i+1}, \mathbf{q}_{i+2}$ responsible for this segment. The quaternion curve can thus be evaluated simply as:

$$\begin{aligned} \boldsymbol{\omega}_i &= \log(\bar{\mathbf{q}}_{i-1}\mathbf{q}_i) \\ \mathbf{q}(\tau) &= \mathbf{q}_{i-1} \exp\left(\boldsymbol{\omega}_1 \widetilde{\mathbf{N}}_1(u)\right) \exp\left(\boldsymbol{\omega}_2 \widetilde{\mathbf{N}}_2(u)\right) \exp\left(\boldsymbol{\omega}_3 \widetilde{\mathbf{N}}_3(u)\right) \end{aligned} \tag{4.4}$$

orientation curve

Note that when evaluated at $u$, $\widetilde{\mathbf{N}}_i(u)$ is a 0-based row vector with 4 entries indexed by $i$. The cumulative B-spline basis has the property that $\widetilde{\mathbf{N}}_0(u) = 1$, which is why we don't need a factor for $\mathbf{q}_{i-1}$. Furthermore, we don't need the exponential mapping here because $\mathbf{q}_{i-2}$ is defined to be the identity rotation, which leads to $\exp(\log(\bar{\mathbf{q}}_{i-2}\mathbf{q}_{i-1}\widetilde{\mathbf{N}}_0(u)) = \mathbf{q}_{i-1}$

### 4.2.3 Orientation interpolation methods

So far we have seen multiple combinations of rotation parameterizations and interpolation methods:

1. Unit quaternion + renormalized LERP (section 3.3)
2. Unit quaternion + piecewise SLERP [Sho85] (section 3.3)
3. Unit quaternion + spherical Bezier curve (SQUAD) [Sho87] (section 3.3)
4. Unit quaternion + cumulative (Bezier,Hermite,B-spline) [KKS95] (section 4.2.2)
5. Infinitesimal rotation + linear interpolation [BZ09, BZF12, KZB16] (section 3.2)
6. Cayley-Gibbs-Rodrigues + basis B-spline [FBS12, FTBS15, ABB14] (section 3.4)
7. SE(3) and se(3) + cumulative uniform cubic B-spline [LPPS13, PPLS15] (section 3.4)
8. Angular (and linear) velocity + uniform cubic B-spline [AB13, AMB15] (section 3.4)
9. Arbitrary Lie Group + cumulative B-spline of arbitrary order [SFSF15] (section 3.4)

We asked ourselves which method suits our needs the best. In [DKL98, Ch. 5], the authors established a framework to visually judge and compare different interpolation methods of orientation keyframes. In particular, they provide a sequence of 6 keyframes (table 4.1) that can nicely be plotted on one hemisphere of the usual sphere $\mathbb{S}^2$. A trajectory in SO(3) can thus be visualized through a curve on the surface of the sphere. Its quality can then be visually judged by evaluating the smoothness of the curve as well as of its derivate, the angular velocity, whose norm is also plotted.

| i | rotation angle $\theta \in [-\pi, \pi]$ | rotation axis $\boldsymbol{\omega} \in \mathbb{R}^3$ | quaternion $\mathbf{q} = [w, x, y, z] \in \mathbb{S}^3$ |
|---|---|---|---|
| 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 4.1: Key frames from [DKL98, Table 5.1]. Assumed that the last rotation axis should be (2,3,0) instead of (1,3,0) to not coincide with the first axis. Note that all z components are chosen to be 0, which allows to drop one dimension and visualize the unit quaternions on the surface of the usual sphere $\mathbb{S}^2$ in $\mathbb{R}^3$, instead of $\mathbb{S}^3$.

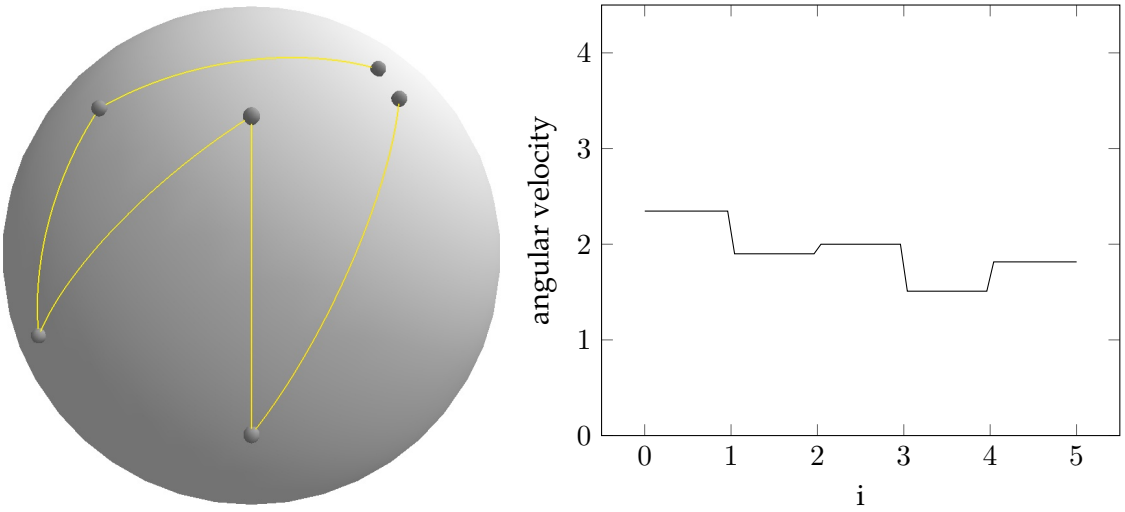



Figure 4.2: Slerp: The keyframes are interpolated piecewise using slerp [Sho85], resulting in local control. However, the curve is only $C^0$ and thus not differentiable at the keyframes. The angular velocity graph is piecewise continuous, meaning that the angular velocity is constant in between keyframes. (The steep lines at the keyframes don't appear vertical here due to numerical computation of the angular velocity using central differences.)

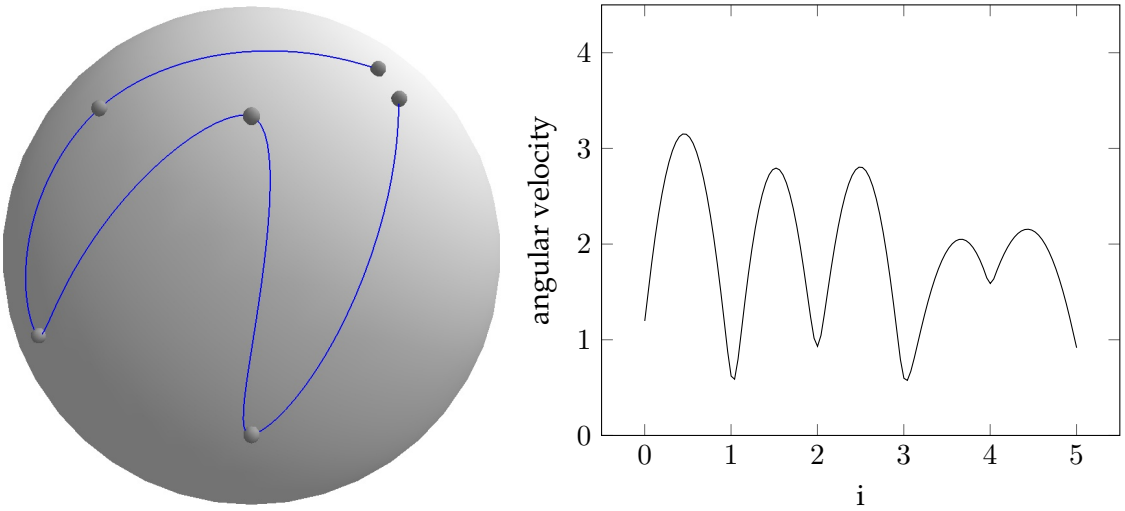



Figure 4.3: Squad: The curve of squad [Sho87] is $C^1$ and thus at least once differentiable everywhere. The angular velocity graph is continuous and has minima at the keyframes. Changes of one control point only propagate to the immediately neighboring segments, thus we have local control.

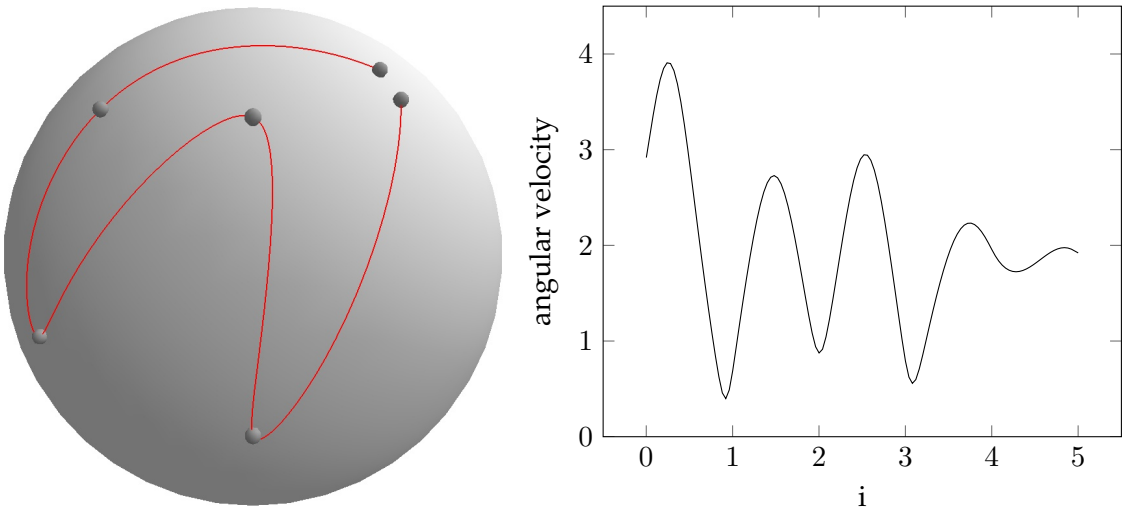



Figure 4.4: Bezier: renormalized. The curve looks very similar to Squad, but is actually a uniform cubic Bézier curve of the quaternion coefficients $\in \mathbb{R}^4$ with subsequent renormalization. The segments are joined with $C^2$ continuity requirements, thus we have global control. Due to the iterated linear interpolation of the quaternion coefficients in the de Casteljau algorithm and subsequent renormalization, the angular velocity between keyframes goes up and then down again. Due to the $C^2$ continuity requirement, these 'wiggles' are more pronounced than in Squad, which is only $C^1$ continuous.

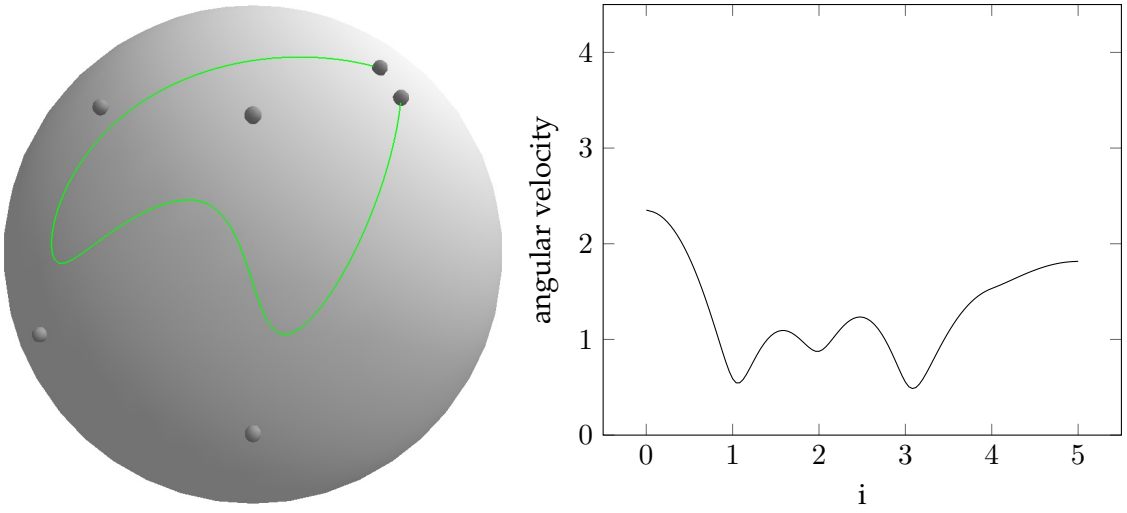



Figure 4.5: B-spline: The curve is $C^2$ and implemented with the uniform cumulative cubic B-spline in $SO(3)$ as introduced by [KKS95] and used in [LPPS13, PPLS15]. Two phantom points are placed so that the curve is defined over the duration of all keyframes while interpolating the first and last one. The angular velocity graph shows that this curve minimizes angular accelerations. Furthermore, it has local control. However, these benefits come at the cost that the inner keyframes are only approximated.

**Summary**

Out of the 9 mentioned methods, we evaluated the four that are based on unit quaternions, since this parameterization is singularity free and only needs one additional parameter compared to minimal representations. Only two of them have $C^2$ continuity (fig. 4.4) (fig. 4.5). Out of these, only one has local control, which is the cumulative cubic B-spline representation for the orientation curve. The fact that it only approximates doesn't matter in optimization (section 4.2.1). Note that the orientation curve is the same for [KKS95] using quaternions and [LPPS13, PPLS15] using transformation matrices, but the latter produces a translation curve with some unwanted properties, as we will see in the next section.

### 4.2.4 Rigid body motion interpolation methods

So far we only looked at interpolating orientations to get a continuous curve $\mathbf{q}(\tau) \in \mathbb{S}^3$ doubly covering $\mathrm{R}(\tau) \in \mathrm{SO}(3)$. However, a rigid body motion additionally consists of a translational part $\mathbf{t} \in \mathbb{R}^3$ that also needs to be interpolated. We introduced two different possible parameterizations for the full rigid body motion (section 2.1.3), namely $(\mathbf{q}, \mathbf{t})$ and $\mathrm{T}$, which differ in the way interpolation for the translational part is carried out.

**Interpolating two keyframes**

Let us first consider just two keyframes:

| i | translation $\mathbf{t}$ | rot. angle $\theta$ | rot. axis $\boldsymbol{\omega}$ | quaternion $\mathbf{q}$ |
|---|---|---|---|---|
| 0 | [0,0,-1] | $\pi/3$ | ( 1, 0, 0) | [0.87, 0.5, 0, 0] |
| 1 | [0,0,1] | $-\pi/3$ | ( 1, 0, 0) | [0.87, -0.5, 0, 0] |

Table 4.2: Full rigid body motion keyframes

**Interpolating $\mathrm{SO}(3)$ and $\mathbb{R}^3$** In Computer Graphics Imaging (CGI), one usually separately interpolates the orientation and translation of the two keyframes, e.G. by applying SLERP (3.9) on $\mathbf{q}_0, \mathbf{q}_1$ and LERP (2.16) on $\mathbf{t}_0, \mathbf{t}_1$:

$$
\begin{aligned}
slerp_{CGI}\big((\mathbf{q}_0, \mathbf{t}_0), (\mathbf{q}_1, \mathbf{t}_1), u\big) &= \big(slerp(\mathbf{q}_0, \mathbf{q}_1, u), lerp(\mathbf{t}_0, \mathbf{t}_1, u)\big) \\
slerp(\mathbf{q}_0, \mathbf{q}_1, u) &= \mathbf{q}_0 \exp(u \log(\bar{\mathbf{q}}_0 \mathbf{q}_1)) \\
lerp(\mathbf{t}_0, \mathbf{t}_1, u) &= \mathbf{t}_0 + u(\mathbf{t}_1 - \mathbf{t}_0)
\end{aligned}
$$

**Interpolating $\mathrm{SE}(3)$** An alternative way to interpolate two rigid body motions is a slerp-like construction in the tangent space $\mathfrak{se}(3)$ of $\mathrm{SE}(3)$:

$$
\begin{aligned}
slerp_{\mathrm{SE}(3)}(\mathrm{T}_0, \mathrm{T}_1, u) &= \mathrm{T}_0 \exp\left(u \log(\mathrm{T}_0^{-1}\mathrm{T}_1)\right) \\
\log(\mathrm{T}_0^{-1}\mathrm{T}_1) &= \log \begin{bmatrix} \mathrm{R}_0^T\mathrm{R}_1 & \mathrm{R}_0^T(\mathbf{t}_1 - \mathbf{t}_0) \\ \mathbf{0}^T & 1 \end{bmatrix}
\end{aligned}
\qquad (4.5)
$$

SE(3) logarithm

However, the interpolation of the translational part does not follow the shortest path between the two keyframes (fig. 4.6). Instead, the intermediate translations somehow depend on the the orientation of the keyframe, as inspection of the $\mathfrak{se}(3)$ tangents reveals: $\mathrm{R}_0^T(\mathbf{t}_1 - \mathbf{t}_0)$ [For14].

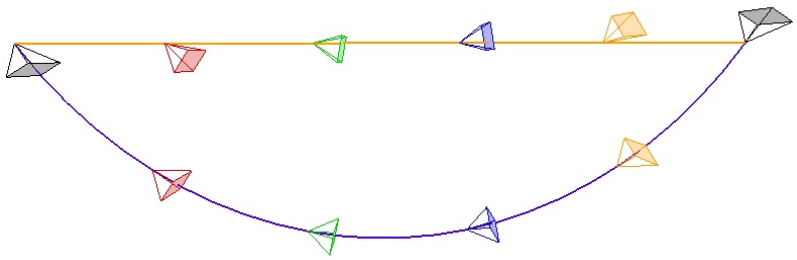


Figure 4.6: The curve for $slerp_{CGI}$ in orange and $slerp_{\mathrm{SE}(3)}$ in violet. Intermediate orientations at equally sampled intervals for both curves are denoted with camera frustums in the same color. Note that the intermediate orientations of the curves are equal, small visual differences are a result of the perspective projection used. The keyframe orientations are black.

#### Trajectory from multiple keyframes with higher order continuity

The above pairwise interpolation schemes can be applied iteratively to gain curves with higher order continuity. In particular, we will focus on the cumulative cubic B-spline, since it provides $\mathrm{C}^2$ continuity and local control.

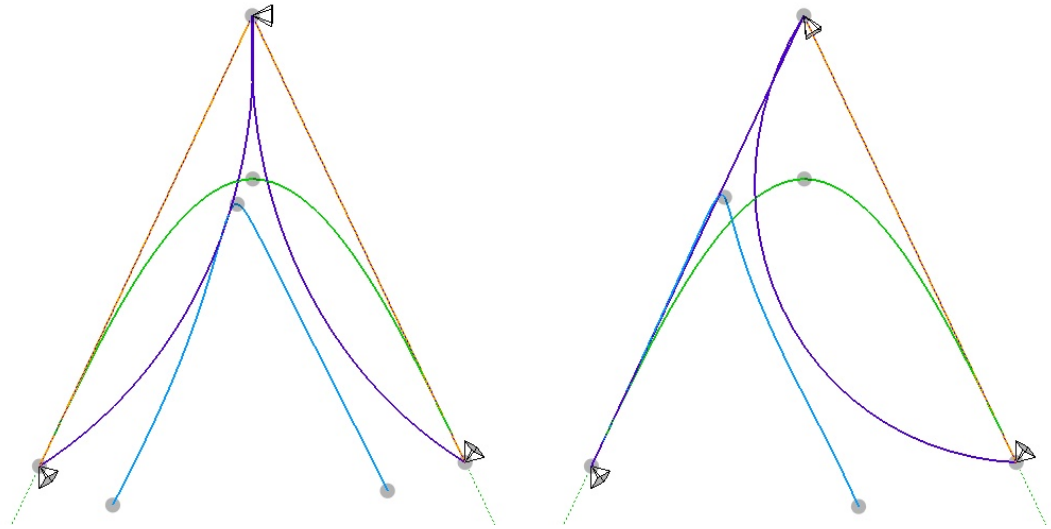
**Trajectory from $\mathrm{SO}(3)$ and $\mathbb{R}^3$** The cumulative cubic B-spline orientation curve (4.4) $\mathbf{q}(\tau)$ construction scheme of [KKS95] can be easily applied to the Euclidean space of translation vectors:

$$\begin{aligned} \mathbf{v}_i &= \mathbf{t}_i - \mathbf{t}_{i-1} \\ \mathbf{t}(\tau) &= \mathbf{t}_0 + \widetilde{\mathbf{N}}_1\mathbf{v}_1 + \widetilde{\mathbf{N}}_2\mathbf{v}_2 + \widetilde{\mathbf{N}}_3\mathbf{v}_3 \end{aligned} \qquad (4.6)$$

translation curve

Thus, through iterated application of $slerp_{CGI}$, we get the independent curves $(\mathbf{q}(\tau), \mathbf{t}(\tau))$.

**Trajectory from $\mathrm{SE}(3)$** Using the same cumulative cubic B-spline but the other parameterization $\mathrm{T} \in \mathrm{SE}(3)$, as used in Spline Fusion [LPPS13, PPLS15] (section 3.4), we get the joint curve $\mathrm{T}(\tau)$. The iterated application of $slerp_{SE(3)}$ leads to a translation curve whose shape is hard to control (fig. 4.7). It is neither bounded by its control polygon, nor does it pass through any control points, even when using phantom points.



(a) The orientation of the intermediate keyframe is set to the identity $\theta = 0 \Rightarrow \mathbf{q}_{0.5} = [1, 0, 0, 0]$, which is exactly halfway between the other two.

(b) The orientation of the intermediate keyframe is set to that of the first keyframe $\mathbf{q}_{0.5} = \mathbf{q}_0$. Note that in this case, $slerp_{CGI} = slerp_{SE(3)}$ for $i \in [0, 0.5]$.

Figure 4.7: Trajectory from three keyframes. The piecewise interpolation of $slerp_{CGI}$ is drawn in orange and $slerp_{SE(3)}$ in violet, the cubic cumulative B-spline trajectory from $\mathrm{SO}(3)$ and $\mathbb{R}^3$ in green and from $\mathrm{SE}(3)$ in cyan. The keyframes are black. Added a new keyframe at $i = 0.5$ with $\mathbf{t}_{0.5} = [0, 2, 0]$ in between the keyframes from the previous example. Phantom points at the beginning and end are added in the direction of the dashed green line so that the first and last control point are interpolated.

### 4.2.5 Summary

The joint curve $\mathrm{T}(\tau)$ might be torque-minimal, but the coupling of orientation and translation is certainly undesired when we need to optimize over trajectories. Thus, we decided to drop this representation.

Instead we chose a trajectory representation that separately interpolates orientation curve (4.4) and translation curve (4.6) to define the combined curve $(\mathbf{q}(\tau), \mathbf{t}(\tau))$. This representation fulfills all the properties we wished for at the beginning of this section:

- *Local control*: an update of the trajectory estimate only affects the trajectory between the neighboring four base poses.
- *$C^2$ continuity of both position and orientation*: through the use of cumulative cubic B-splines for both.
- *Orientation parameterization without singularities*: through the use of unit quaternions.
- *As few parameters as possible*: through the use of unit quaternions and a medium knot spacing.
- *Analytic derivatives with efficient formulas*: The computation of derivatives on B-splines and quaternions is actually not hard (section 4.4.2).

## 4.3 Algorithm Overview

This section gives an overview of the algorithm at the core of our approach. We explain each line of the pseudo-code and give references to further details.

**Algorithm 1:** CONTINUOUS TIME TRAJECTORY ESTIMATION FOR 3D SLAM FROM AN ACTUATED 2D LASER SCANNER

**Input**: $\mathcal{L}_{\tau_k}$ 2D laser scans at discrete times $\tau_k$
$\mathcal{M}_{\tau_k}$ IMU measurements at discrete times $\tau_k$
**Output**: $T(\tau)$ continuous time trajectory
**Params**: $\Delta\tau_{Sweep}$

1 $T \leftarrow initializeTrajectoryFromIMU(\mathcal{M})$
2 **for** $i \leftarrow 1$, $\tau_{curr} \leftarrow \min_\tau(\mathcal{L})$ ***to*** $\max_\tau(\mathcal{L})$; $i \leftarrow i+1$, $\tau_{curr} \leftarrow \tau_{curr} + \Delta\tau_{Sweep}$ **do**
3 $\quad S_i \leftarrow unprojectAndAccumulate([\mathcal{L}_{\tau_{curr}}, \ldots, \mathcal{L}_{\tau_{curr}+\Delta\tau_{Sweep}}])$
4 $\quad S_i \leftarrow undistortAndComputeSurfels(S_i, T)$
5 $\quad$ **if** $i > 1$ **then**
6 $\quad\quad$ imuMatches $\leftarrow imuDeviations(T, [\mathcal{M}_{\tau_{curr}-\Delta\tau_{Sweep}}, \ldots, \mathcal{M}_{\tau_{curr}+\Delta\tau_{Sweep}}])$
7 $\quad\quad$ **for** $j \leftarrow 1$ ***to*** $n_{icpIter}$ **do**
8 $\quad\quad\quad$ surfelMatches $\leftarrow matchSurfels(S_i(T), S_{i-1}(T))$
9 $\quad\quad\quad T \leftarrow NLS(T,$ surfelMatches, imuMatches$)$
10 $\quad\quad\quad S_i \leftarrow updateSurfelWorldPosition(S_i, T)$
11 **return** $T(\tau)$

The input to our algorithm are 2D laser scans and IMU measurements. It operates by advancing a time-window (line 2) by half of its length, the sweep duration $\Delta\tau_{Sweep}$. For each pair of sweeps (line 5) the two basic ICP steps are repeated until convergence (line 7):
I compute **correspondences** (line 8) and
II apply the **transformation** (line 10) to minimize the error of these matches.

The trajectory is initialized (line 1) using IMU measurements (section 4.4.1). The 2D laser scans are converted to Cartesian coordinates and accumulated (line 3) into a sweep (section 4.5.1). With the current trajectory estimate we can (line 4) undistort all scans in a sweep, assign true world coordinates and compute surfels (section 4.5.2) by discretizing space into a voxel grid.

Trajectory regularization constraints (line 6) are computed once for the current time window (section 4.4.3). For a pair of sweeps, we subsequently solve multiple NLS problems (line 9) in an ICP fashion. Multiple observations of the same landmark are denoted as surfel matches (line 8) and need to be recomputed after each ICP iteration (section 4.5.3).

Using the sliding time-window approach of above algorithm for each pair of consecutive sweeps we get an open-loop trajectory estimate (section 4.6.1). With minor modifications, the same algorithm is employed subsequently to globally register all sweeps at once to further reduce the accumulated drift (section 4.6.2).

## 4.4 Using IMU measurements

To get an initial estimate of the trajectory, we integrate IMU measurements (section 4.4.1) using inertial navigation techniques. Furthermore, we want to incorporate IMU measurements constraints in our optimization problem to act as a regularizer for the surfel matches (section 4.5.3). To do so, we must first derive analytical expressions (section 4.4.2) for the angular velocity and linear acceleration of our trajectory representation. Then we may state the residual terms, taking care of the noise model of the measured data (section 4.4.3).

### 4.4.1 Trajectory initialization via inertial navigation

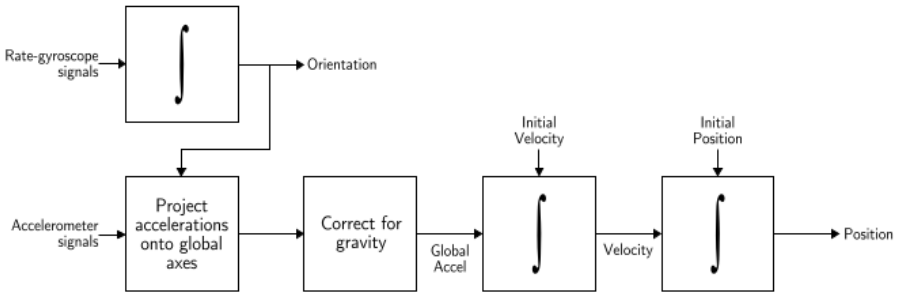



Figure 4.8: Strapdown inertial navigation algorithm [Woo07, Fig 4]

*Inertia* is the property of a rigid body to maintain constant linear and angular velocity, unless disturbed by forces or torques, respectively. This is valid in an *inertial reference frame* where Newton's laws of motion hold. When ignoring the negligible effects due to the earth's rotation, the world frame $_w$ can be seen as such. For the time derivative of a rigid body motion $g = (\mathrm{R}, \mathbf{t})$ or $g = (\mathbf{q}, \mathbf{t})$, it holds that [FCDS15, KKS95]:

$$\dot{\mathrm{R}} = \mathrm{R}[\boldsymbol{\omega}]_\times \text{ or } \dot{\mathbf{q}} = \mathbf{q}\frac{1}{2}\boldsymbol{\omega} \quad , \quad \dot{\mathbf{t}} = \mathbf{v} \text{ and } \ddot{\mathbf{t}} = \dot{\mathbf{v}} = \mathbf{a}$$

When knowing $\boldsymbol{\omega}(\tau)$ and $\mathbf{a}(\tau)$ as well as the initial velocity and position, we can recover the rigid body motion $g$, consisting of position $\mathbf{t}(\tau)$ and orientation $\mathrm{R}(\tau)$ or $\mathbf{q}(\tau)$ via simple numerical integration [FCDS15]:

$$\begin{aligned}
\mathbf{q}(\tau + \Delta\tau) &= \mathbf{q}(\tau)\exp(\boldsymbol{\omega}(\tau)\Delta\tau) \\
{}_w\mathbf{v}(\tau + \Delta\tau) &= {}_w\mathbf{v}(\tau) + {}_w\mathbf{a}(\tau)\Delta\tau \\
{}_w\mathbf{t}(\tau + \Delta\tau) &= {}_w\mathbf{t}(\tau) + {}_w\mathbf{v}(\tau)\Delta\tau + \frac{1}{2}{}_w\mathbf{a}(\tau)\Delta\tau^2
\end{aligned}$$

*Inertial navigation* tries to track the position and orientation of an object by processing and integrating noisy IMU measurements angular velocity (2.14) $\tilde{\boldsymbol{\omega}}$ and linear acceleration (2.15) $\tilde{\mathbf{a}}$, sampled at a specific frequency $\Delta\tau$, instead of the unavailable true functions $\boldsymbol{\omega}$ and $\mathbf{a}$. In strapdown inertial navigation, $\tilde{\mathbf{a}}$ is not measured in the frame $_w$, but in $_s$, which makes things more complicated. Furthermore, $\mathbf{a}(\tau)$ and $\boldsymbol{\omega}(\tau)$ most likely don't remain

constant in $[\tau, \tau + \Delta\tau]$, which additionally introduces integration errors. For $\tilde{\boldsymbol{\omega}}$, there is nothing we can do, but for $\tilde{\mathbf{a}}$, we can do slightly better using the midpoint rule:

$$
\begin{aligned}
{}_w\mathbf{a}(\tau + \tfrac{1}{2}\Delta\tau) &= \frac{1}{2}\left(\mathbf{q}(\tau){}_s\mathbf{a}(\tau) + \mathbf{q}(\tau + \Delta\tau){}_s\mathbf{a}(\tau + \Delta\tau)\right) \\
{}_w\mathbf{v}(\tau + \Delta\tau) &= \mathbf{q}(\tau){}_s\mathbf{v}(\tau) + {}_w\mathbf{a}(\tau + \tfrac{1}{2}\Delta\tau)\Delta\tau \\
{}_s\mathbf{v}(\tau + \Delta\tau) &= \bar{\mathbf{q}}(\tau + \Delta\tau){}_w\mathbf{v}(\tau + \Delta\tau) \\
{}_w\mathbf{t}(\tau + \Delta\tau) &= {}_s\mathbf{t}(\tau) + {}_w\mathbf{v}(\tau + \Delta\tau)\Delta\tau
\end{aligned}
$$

Due to the double integration of acceleration and its dependency on accurate orientation, translation errors are usually much larger than orientation errors. Rotation rates only need to be integrated once and are independent of acceleration. The non-zero centered bias introduces an additional drift in pose estimation, which makes estimation of a long trajectory (even 1s is already long) unfeasible if only IMU data is used.

Because of these shortcomings we only use inertial navigation as a first guess for trajectory initialization, which is later on refined in combination with other techniques. However, over short time-spans and because of its high sensor rate, IMU data provide useful time-local constraints which can be used for trajectory regularization as seen in the next section.

### 4.4.2 Synthesized inertial measurements

We decided to use the cumulative cubic B-spline approach of [KKS95] (section 4.2.2) for the orientation curve (4.4) $\mathbf{q}(\tau)$. The same construction scheme was also applied for the translation curve (4.6) $\mathbf{t}(\tau)$ so that we can use the same cubic cumulative basis matrix $\widetilde{\mathbf{N}}(\tau)$. For simplification of notation, we omit the time variable $(\tau)$ from the basis matrix and just write $\widetilde{\mathbf{N}}$. Furthermore, we index the control points or quaternions of the current segment with $0, 1, 2, 3$ instead of $i-1, i, i+1, i+2$.

The derivative of a B-spline curve with respect to time is easily computed. The only thing that depends on time is our local parameter $u$, which appears in polynomial form up to degree 3. Thus we can split the basis matrix $\widetilde{\mathbf{N}}$ into a row vector of polynomial coefficients and a constant matrix $\mathbf{C}$. The derivatives of a B-spline just need to use the derivatives of the B-spline basis matrix $\dot{\widetilde{\mathbf{N}}}$ and $\ddot{\widetilde{\mathbf{N}}}$ and furthermore just follow the chain rule. Note that $\widetilde{\mathbf{N}}_0 = 1, \dot{\widetilde{\mathbf{N}}}_0 = 0, \ddot{\widetilde{\mathbf{N}}}_0 = 0$:

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

Because the orientation curve $\mathbf{q}(\tau)$ is given as a (quaternion) product with 3 non-constant factors $x_i = \exp(\widetilde{\mathbf{N}}_i\boldsymbol{\omega}_i)$ its first time derivative $\dot{\mathbf{q}}(\tau)$ looks a little lengthy. However, it is

just given by iterated application of product rule $(x_1x_2x_3)' = x_1'x_2x_3 + x_1x_2'x_3 + x_1x_2x_3'$ and chain rule for each summand $x_i' = (u_i(v_i))' = u_i'(v_i)v_i'$. With $u_i = \exp(v_i)$ and $v_i = \widetilde{\mathbf{N}}_i\boldsymbol{\omega}_i$ we get $x_i' = \exp(\widetilde{\mathbf{N}}_i\boldsymbol{\omega}_i)' = \exp(\widetilde{\mathbf{N}}_i\boldsymbol{\omega}_i)(\dot{\widetilde{\mathbf{N}}}_i\boldsymbol{\omega}_i)$. The anchor control point $\mathbf{q}_0$ is just a constant factor, since it does not depend on time.

$$
\begin{aligned}
\boldsymbol{\omega}_i &= \log(\bar{\mathbf{q}}_{i-1}\mathbf{q}_i) \\
\mathbf{q}(\tau) &= \mathbf{q}_0 \exp(\widetilde{\mathbf{N}}_1\boldsymbol{\omega}_1) \exp(\widetilde{\mathbf{N}}_2\boldsymbol{\omega}_2) \exp(\widetilde{\mathbf{N}}_3\boldsymbol{\omega}_3) \\
\dot{\mathbf{q}}(\tau) &= \mathbf{q}_0 \exp(\widetilde{\mathbf{N}}_1\boldsymbol{\omega}_1)(\dot{\widetilde{\mathbf{N}}}_1\boldsymbol{\omega}_1) \exp(\widetilde{\mathbf{N}}_2\boldsymbol{\omega}_2) \exp(\widetilde{\mathbf{N}}_3\boldsymbol{\omega}_3) \\
&\quad + \mathbf{q}_0 \exp(\widetilde{\mathbf{N}}_1\boldsymbol{\omega}_1) \exp(\widetilde{\mathbf{N}}_2\boldsymbol{\omega}_2)(\dot{\widetilde{\mathbf{N}}}_2\boldsymbol{\omega}_2) \exp(\widetilde{\mathbf{N}}_3\boldsymbol{\omega}_3) \\
&\quad + \mathbf{q}_0 \exp(\widetilde{\mathbf{N}}_1\boldsymbol{\omega}_1) \exp(\widetilde{\mathbf{N}}_2\boldsymbol{\omega}_2) \exp(\widetilde{\mathbf{N}}_3\boldsymbol{\omega}_3)(\dot{\widetilde{\mathbf{N}}}_3\boldsymbol{\omega}_3) \\
{}_s\boldsymbol{\omega}(\tau) &= 2(\bar{\mathbf{q}}(\tau)\dot{\mathbf{q}}(\tau)).vec()
\end{aligned}
$$

The correct formula for the angular velocity curve $\boldsymbol{\omega}(\tau) = 2(\bar{\mathbf{q}}(\tau)\dot{\mathbf{q}}(\tau)).vec()$ is not trivially derived. It did not appear in any previous work that we know of and thus deserves an in-depth explanation. $\dot{\mathbf{q}}(\tau)$ is given in the world frame ${}_w$. By multiplication with the inverse $\bar{\mathbf{q}}(\tau)$ of the orientation curve $\mathbf{q}(\tau)$, we transform it into the sensor frame ${}_s$. The stunning fact is that ${}_s\dot{\mathbf{q}}(\tau) = \bar{\mathbf{q}}(\tau){}_w\dot{\mathbf{q}}(\tau)$, the product of two unit quaternions from the Lie group $\mathrm{SU}(2)$, becomes a pure quaternion (with ${}_s\dot{q_w}(\tau) = 0$) in the Lie algebra $\mathfrak{su}(2)$! This is due to the fact that in the sensor frame ${}_s$, the pure quaternion can be seen as a representative of the tangent space at the identity. We can identify the vector part ${}_s\dot{\mathbf{q}}_{x:z}(\tau)$ of this pure quaternion with an angular velocity vector $\in \mathbb{R}^3$. The multiplication by two may be needed since the curve's velocity is measured in $\mathbb{S}^3$, but the angular velocity in $\mathrm{SO}(3)$. This depends however how one defines quaternion exp (2.5). Fortunately we only need the first derivative of the orientation curve to get synthetic analytic angular velocity measurements $\boldsymbol{\omega}$ for an arbitrary time $\tau$. To synthesize linear acceleration measurements, we need to derive the translation curve (4.6) $\mathbf{t}(\tau)$ twice. However it is much easier to derive because it is just a sum so each summand can be derived independently. Each summand is furthermore just a product where $\mathbf{v}_i$ is a constant factor, so there is no need for the chain rule either. This makes computing the second derivative trivial:

$$
\begin{aligned}
\mathbf{v}_i &= \mathbf{t}_i - \mathbf{t}_{i-1} \\
{}_w\mathbf{t}(\tau) &= \mathbf{t}_0 + \widetilde{\mathbf{N}}_1\mathbf{v}_1 + \widetilde{\mathbf{N}}_2\mathbf{v}_2 + \widetilde{\mathbf{N}}_3\mathbf{v}_3 \\
{}_s\mathbf{v}(\tau) &= \bar{\mathbf{q}}(\tau)\big(\dot{\widetilde{\mathbf{N}}}_1\mathbf{v}_1 + \dot{\widetilde{\mathbf{N}}}_2\mathbf{v}_2 + \dot{\widetilde{\mathbf{N}}}_3\mathbf{v}_3\big) = {}_s\dot{\mathbf{t}}(\tau) \\
{}_s\mathbf{a}(\tau) &= \bar{\mathbf{q}}(\tau)\big(\ddot{\widetilde{\mathbf{N}}}_1\mathbf{v}_1 + \ddot{\widetilde{\mathbf{N}}}_2\mathbf{v}_2 + \ddot{\widetilde{\mathbf{N}}}_3\mathbf{v}_3\big) = {}_s\dot{\mathbf{v}}(\tau)
\end{aligned}
$$

Note that this formula is much easier than the matrix-based formulas presented in Spline Fusion [LPPS13, PPLS15]. This is due to the fact that our position and orientation curve are independent of each other. Easier formulas mean more efficient optimization when using Ceres automatic derivatives on the residuals.

### 4.4.3 Trajectory regularization constraints

The measured angular velocities (2.14) and linear accelerations (2.15) follow the following noise model as explained before:

$$
\begin{aligned}
{}_s\tilde{\boldsymbol{\omega}}(\tau) &= {}_s\boldsymbol{\omega}(\tau) + {}_s\mathbf{b}_{\boldsymbol{\omega}}(\tau) + \mathcal{N}(\mathbf{0}, \sigma_{\boldsymbol{\omega}}) \\
{}_s\tilde{\mathbf{a}}(\tau) &= {}_s\mathbf{a}(\tau) + \bar{\mathbf{q}}(\tau)_w\mathbf{g} + {}_s\mathbf{b}_{\mathbf{a}}(\tau) + \mathcal{N}(\mathbf{0}, \sigma_{\mathbf{a}})
\end{aligned}
$$

The residual between a synthesized and a real IMU measurement, consisting of gyroscope and accelerometer readings at time $\tau_k$ may be written as:

$$
{}_s\mathbf{e}_{\boldsymbol{\omega}} = {}_s\tilde{\boldsymbol{\omega}}(\tau_k) - {}_s\boldsymbol{\omega}(\tau_k) - {}_s\mathbf{b}_{\boldsymbol{\omega}}(\tau_k) \tag{4.7}
$$

angular velocity residual

$$
{}_s\mathbf{e}_{\mathbf{a}} = {}_s\tilde{\mathbf{a}}(\tau_k) - \bar{\mathbf{q}}(\tau_k)_w\mathbf{g} - {}_s\mathbf{a}(\tau_k) - {}_s\mathbf{b}_{\mathbf{a}}(\tau_k) \tag{4.8}
$$

acceleration residual

Note that in contrast to (3.8), we state the acceleration residual (4.8) in the sensor frame instead of the world frame. This allows to directly substract the synthesized acceleration measurement ${}_s\mathbf{a}$ from the previous section as well as the bias ${}_{ss}\mathbf{b}$. Only the gravity has to be rotated first. Due to the zero-centered white noise on the measurements, the residuals should be weighted by their respective information matrices $\Sigma^{-1}$, obtained from device specifications or calibration. These allow weighing the expected variance of each dimension against each other. The bias functions are modeled as as cumulative cubic B-spline and share the same knots as our control points. To keep them constrained, we furthermore add a term for each measurement time forcing the bias' derivative to zero:

$$
\mathbf{e}_{{}_s\mathbf{b}_{\boldsymbol{\omega}}} = {}_s\dot{\mathbf{b}}_{\boldsymbol{\omega}}(\tau_k) \quad , \quad \mathbf{e}_{{}_s\mathbf{b}_{\mathbf{a}}} = {}_s\dot{\mathbf{b}}_{\mathbf{a}}(\tau_k) \tag{4.9}
$$

bias regularizer residual

The derivative of the B-spline curve ${}_s\dot{\mathbf{b}}(\tau)$ can be computed in analogy to the first derivative of our position curve, the velocity curve $\dot{\mathbf{t}}(\tau) = \mathbf{v}(\tau)$. The bias control points are initialized with a zero vector.

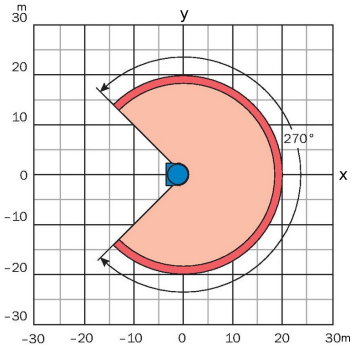
In contrast to the bias, the gravity ${}_w\mathbf{g}$ vector is a constant and measured in the world frame. It is initialized by taking the mean of acceleration measurements while leaving the device fixed and level with the floor plane for a few seconds. Normally, the world frame used in IMU's is the North East Down frame, with the gravity vector pointing towards negative z. In an ideal world the mean of acceleration measurements when leaving the device fixed should be $[0, 0, 9.81]$. Due to magnetic interference, inexact orientation measurements or internal calibration errors of the 3 accelerometer's axis, even when the device stays fixed, parts of the gravity's z component will be projected onto the other axes. Thus, it could make sense to model the gravity not as a constant, but as a slowly time-varying function in the world frame, similar to the bias. However, we leave this for future work.

Alternatively, some devices allow to compensate for gravity directly on the device. In that case we can drop the term $\bar{\mathbf{q}}(\tau_k)_w\mathbf{g}$ above.

## 4.5 Using LiDAR measurements

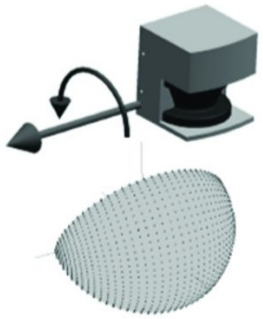
Our algorithm does not use the 2D LiDAR measurements directly, but accumulates them into a sweep first (section 4.5.1). It then computes local surface features, surfels on this 3D representation (section 4.5.2) which are used as landmarks. Multiple observations of the same landmark, named *surfel matches* are used as constraints in optimization (section 4.5.3).

### 4.5.1 Accumulating 2D laser scans into a 3D sweep




(a) 2D laser rotating around the yaw or z-axis generates measurements in the x-y-plane. It has a field of view of 270 degrees and a range of about 20 m (fig. 2.2b).

(b) Rotation around the roll or x-axis allows to cover the environment visible to the sensor in every half rotation due to the symmetry along the x-axis. Points to the front are most densely sampled.



(c) Nodding along the pitch or y-axis is also sometimes encountered. However, it needs twice as long as rolling motion to cover the whole environment. Points to the front are least densely sampled.

Figure 4.9: 2D laser scan plane and possible actuation [WW03] to get a 3D scan.

A 2D laser scanner rotates its mirror around the z-axis (fig. 4.9a) to get measurements in the x-y-plane in polar coordinates. These are converted to Cartesian coordinates using lidar backprojection (2.13). To get 3D measurements from a 2D sensor, we have to actuate it to cover the remaining dimension and then fuse multiple scans.

For the actuation, there are two possibilities. We made use of symmetry along the x-axis by rotating the 2D laser scanner around it (fig. 4.9b). This allows to cover the whole space visible to the scanner in half the timespan than when rotating or nodding around the y-axis (fig. 4.9c) with the same frequency. To fuse multiple scans we have to take into account the motion of the scanner, otherwise the accumulated scans will have distortion artefacts due to the intra- and inter-scan motion. Thus, we need the trajectory of the laser scanner during the acquisition of a scan, an initial estimate of which is obtained via inertial navigation (section 4.4.1).

We can account for intra and inter scan distortion in a unified manner if we treat each single laser beam individually. We know the acquisition time $\tau_k$ of each laser beam and can query our trajectory estimate $T$ for the current transformation between sensor and world frame. W.l.o.g lets assume that the laser scanner origin coincides with the IMU sensor frame ${}_s$. When seen in this varying frame, the simple projection from polar to Cartesian coordinates is actually distortion free and results in a point ${}_s\mathbf{p} = [x, y, 0]$ having a z component of zero. To get the correct coordinates of each laser beam in the world frame, we just have to apply: ${}_w\mathbf{p} = T(\tau_k){}_s\mathbf{p}$

Depending on the frequency of the actuation and of the laser scanner mirror rotation we gather a certain number of 2D laser scans into a 3D sweep. E.g. if actuated with 0.5Hz, one half rotation lasts 1s. If the laser scanner rotates with 40Hz, we get 40 laser scans per sweep.

### 4.5.2 Generating surfels

Once all the scans of a sweep are projected into the world frame, we can compute surfels. This is done by discretizing the space covered by a sweep into a 3D voxel-grid. If enough points fall inside a voxel, the PCA of all the points inside this voxel is computed. This fits a Gaussian distribution to the point set inside a voxel and summarizes it using the centroid (2.21) and the covariance matrix (2.22), whose eigen decomposition (2.23) is also computed to gain simple surface statistics. Based on the sorted eigenvalues $\lambda_0 \leq \lambda_1 \leq \lambda_2$, we are computing the plane likeness $\in [0, 1]$ to measure to judge the local shape inside the voxel as follows [BZ09]:

$$P(\{\mathbf{p}_i\} \text{ describes plane}) = 2\frac{\lambda_1 - \lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{4.10}$$

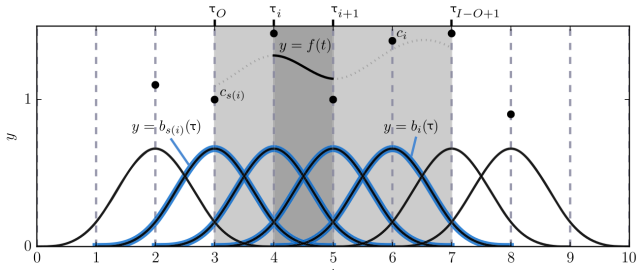
plane-likeness

The advantage of this measure compared to curvature (2.24) is that it gives special attention to the second smallest eigenvalue $\lambda_1$ and not only to the smallest eigenvalue $\lambda_0$ as the curvature does. It gives a value of 1, indicating a planar region, only if the smallest eigenvalue is significantly smaller then the other two. The curvature, on the other hand, cannot distinguish between planar and cylindrical features, where the second smallest eigenvalue is about the same size as the smallest eigenvalue. These cylindrical features can appear in point cloud acquisition with LiDAR when a voxel is traversed by the scan-line only once. Cylindrical features cannot be used to compute a reliable 3D normal, since the normal can rotate around the line as long as it stays perpendicular to it. The curvature measure was originally created for dense point clouds acquired by range cameras, where these cylindrical features did not appear.

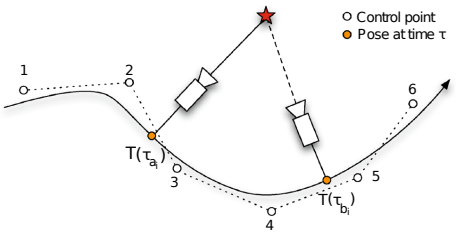
However, if the plane-likeness (4.10) is high enough, we can compute the surface normal (2.25) $\vec{\mathbf{n}}$ and generate a surfel. All the surfels of a sweep are now put into a 6D kd-tree for fast nearest neighbor queries, where the 6D vector $[\boldsymbol{\mu}, \vec{\mathbf{n}}]^T$ is the concatenation of centroid and normal vector.

### 4.5.3 Surfel match constraints

Multiple observations of the same landmark (fig. 4.10b) in consecutive sweeps are approximated by nearest neighbor surfel matches in the 6D Euclidean space of concatenated centroid and normal vectors. Only reciprocal matches are kept to keep the match error between two sweeps symmetric. Matches whose distance between their centroids or angle between surface normals is above a predefined threshold are also discarded. This allows to get rid of wrong initial correspondences during later ICP iterations. However, the thresholds should not be set too low initially, since it is possible that one then discards all correspondences and gets stuck in a local minima. An adaptive threshold that is initially




(a) Graph of a 1D cubic B-spline. The bumps are the basis functions, the knots are the dashed lines, while the control vertices are the black dots. The curve is only defined inside the shaded area, where 4 basis functions are nonzero. Note that 2 phantom control points outside this area are needed to define the curve. The shape of the curve in the dark gray area is influenced only by the highlighted basis functions. [SFSF15]




(b) A surfel match is a pair of landmark observations at different times along the trajectory. The pose of surfel $a_i$ at time $\tau_{a_i}$ is parameterized by control poses 1-4 and the pose of surfel $b_i$ by control poses 3-6. Due to the overlap of two, this surfel match $\{a_i \rightarrow b_i\}$ constrains 6 control poses. [PPLS15]

Figure 4.10: Surfel matches and the resulting constraints on the trajectory representation.

set high to allow for a larger basin of convergence and then reduced at each iteration to allow for a high accuracy final solution is thus preferable.

By stating GICP's plane to plane (3.6) relative pose error:

$$E = \sum_{i=1}^{N} d(\mathbf{p}_i, \mathbf{p}'_i)^T (\mathrm{R}\hat{\Sigma}_{\mathbf{p}_i}\mathrm{R}^T + \hat{\Sigma}_{\mathbf{p}'_i})^{-1} d(\mathbf{p}_i, \mathbf{p}'_i) \quad , \quad d(\mathbf{p}_i, \mathbf{p}'_i) = \mathrm{R}\mathbf{p}_i + \mathbf{t} - \mathbf{p}'_i$$

as an absolute pose error (3.3) like in the derivation of MV-LM-ICP:

$$E(g_h, g_k) = \sum_{i=1}^{N_h} \left\| \mathrm{R}_h\mathbf{p}_i + \mathbf{t}_h - (\mathrm{R}_k\mathbf{p}'_i + \mathbf{t}_k) \right\|^2$$

And replacing $N$ point correspondences $\{\mathbf{p}_i \rightarrow \mathbf{p}'_i\}_1^N$ (section 3.1) with one surfel match $a_i \rightarrow b_i$, where each surfel is a tuple of centroid, normal, and covariance matrix with disc shape prior, $(\boldsymbol{\mu}, \vec{\mathbf{n}}, \hat{\Sigma})$, we get an expression for the residual:

$$\begin{aligned} \bar{d}(a_i, b_i) &= \mathrm{R}_{a_i}\boldsymbol{\mu}_{a_i} + \mathbf{t}_{a_i} - (\mathrm{R}_{b_i}\boldsymbol{\mu}_{b_i} + \mathbf{t}_{b_i}) \\ \mathbf{e}_{a_i,b_i} &= \bar{d}(a_i, b_i)^T (\mathrm{R}_h\hat{\Sigma}_{a_i}\mathrm{R}_h^T + \mathrm{R}_k\hat{\Sigma}_{b_i}\mathrm{R}_k^T)^{-1} \bar{d}(a_i, b_i) \end{aligned} \tag{4.11}$$

surfel match residual

This is the plane to plane error using absolute poses adapted for use with surfels, whose vector- or matrix-valued quantities $(\boldsymbol{\mu}, \vec{\mathbf{n}}, \hat{\Sigma})$ are expressed in the sensor frame $_s$. The error above however is expressed in the world frame $_w$, by first rotating and translating the centroid and rotating the covariance matrices using the rigid body motions $(\mathbf{q}_{a_i}, \mathbf{t}_{a_i}) = T(\tau_{a_i}), (\mathbf{q}_{b_i}, \mathbf{t}_{b_i}) = T(\tau_{b_i})$ acquired by evaluating the trajectory $T(\tau)$ at the surfel's mean timestamp.

A surfel match constrains the trajectory at two instants in time. Due to the parametrization with cubic Bezier splines (fig. 4.10a), between 4 and 8 distinctive base poses are affected by this match. This is the case because the influence of the basis functions at the two specific times may overlap.

## 4.6 Model optimization

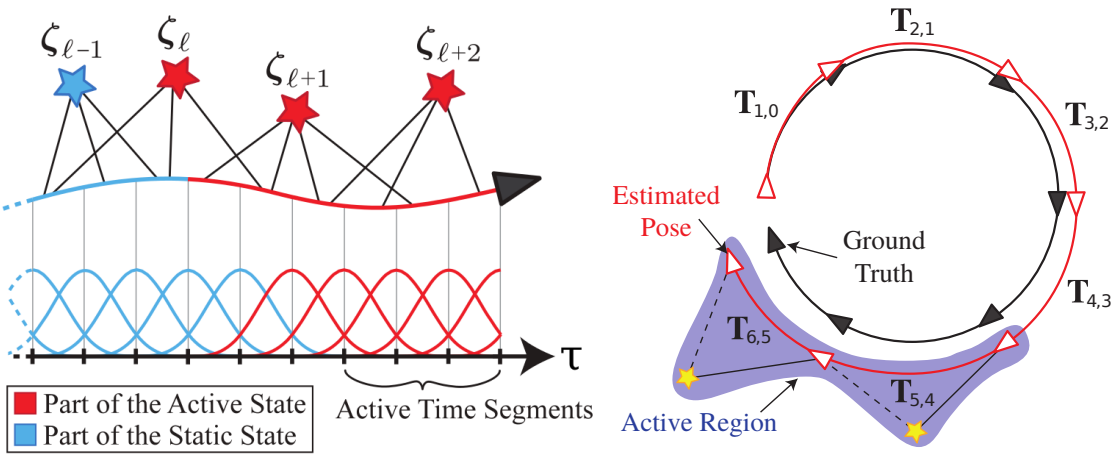
The overall objective function to be minimized is the sum of squared norms of all our residuals:

$$E = (1-\alpha)\sum_{i=0}^{N} \mathbf{e}_{a_i,b_i} + \alpha \sum_{j=1}^{M} \left( \mathbf{e}_{\boldsymbol{\omega}_j}^T \Sigma_{\boldsymbol{\omega}}^{-1} \mathbf{e}_{\boldsymbol{\omega}_j} + \mathbf{e}_{\mathbf{a}_j}^T \Sigma_{\mathbf{a}}^{-1} \mathbf{e}_{\mathbf{a}_j} + \mathbf{e}_{s\mathbf{b}_{\boldsymbol{\omega}}}^T \mathbf{e}_{s\mathbf{b}_{\boldsymbol{\omega}}} + \mathbf{e}_{s\mathbf{b}_{\mathbf{a}}}^T \mathbf{e}_{s\mathbf{b}_{\mathbf{a}}} \right)$$

We have $N$ surfel matches, each contributing a surfel match residual (4.11). Note that in the way we defined this residual, it is already a scalar, a squared norm weighted by an information matrix, the surface shape prior. Each of the $M$ IMU measurements contributes an angular velocity residual (4.7) and an acceleration residual (4.8) together with a bias regularizer residual (4.9) for each of them. These are vector valued quantities $\in \mathbb{R}^3$ and thus their norm needs to be taken. The measurements are furthermore weighted by their information matrices.

We introduce the scalar $\alpha \in [0, 1]$, which allows to weigh the contributions of these two error terms against each other. It acts as a regularization parameter controlling the weight of the data term arising from surfel matches vs. the regularization term consisting of IMU constraints. Setting $\alpha$ to zero has the effect of tearing trajectories apart, since unbounded accelerations are no longer punished. Setting it to one allows the above algorithm to work as a sophisticated inertial navigation algorithm which uses splines for integration.

We solve the objective function via Non-linear Least Squares (NLS). It is quite general, and can thus be employed in two operation modes, which only differ by the way the residual terms are added. We can either process the trajectory in small subsets, which we coin *time-windowed optimization* (section 4.6.1) or in batch, that we refer to as *global optimization* (section 4.6.2).




(a) The active state during the time-windowed optimization. Uniform cubic B-spline basis functions that influence the active time segments are part of the active state. A surfel match, denoted with a star, becomes active if at least one of the surfels is from within the active state.

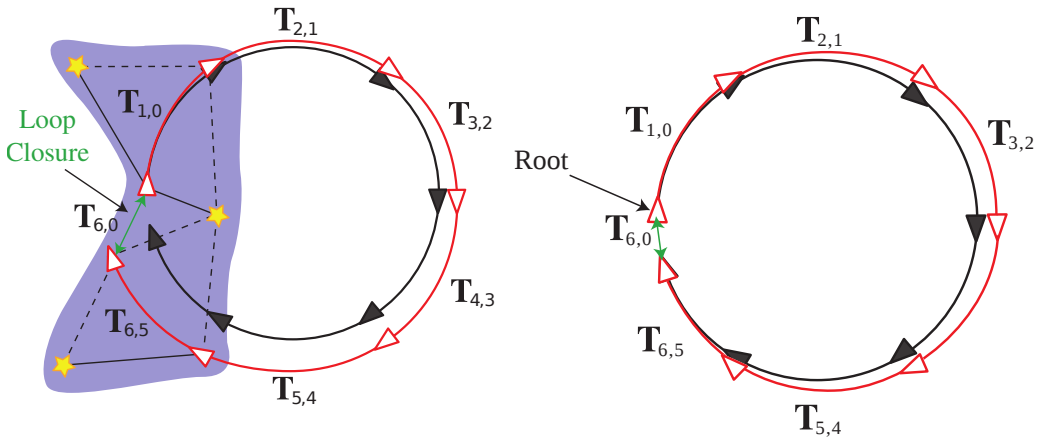
(b) In the case of time-windowed optimization, the active state usually spans a few of the most recent base poses. Note that drift accumulates due to the frame-to-frame estimation.

Figure 4.11: Time-windowed or open loop trajectory estimation works by optimizing a small subset of the trajectory at a time and advancing the window as new data arrives [AMB15].

### 4.6.1 Time-windowed optimization

Our algorithm is applied for open-loop trajectory estimation in a time-windowed fashion. In the first iteration, the active state comprises of all the base poses of the first two sweeps. Then we iterate by advancing the current time window by a fraction of its length, usually the duration of one sweep. Now only the base poses of the latest sweep are optimized, the others are static (fig. 4.11a). This ensures that the error of an already processed segment does not increase again. However, this frame-to-frame or sweep-to-sweep approach is inherent to accumulate drift (fig. 4.11b).

### 4.6.2 Global optimization




(a) In the case of loop closure, the active state consists of surfels and base poses from both ends of the trajectory. Drift between the ends can be corrected for.

(b) When closing the loop, the remaining error is evenly distributed along the rest of the trajectory through global optimization.

Figure 4.12: Loop closures and global optimization improve the final accuracy [AMB15].

After all data is processed, we have an open-loop trajectory estimate in which drift from discretization errors is accumulated. To reduce the drift, we apply a subsequent global optimization step, optimizing over all sweeps at once. If the open-loop trajectory estimate is good enough, loop closures will automatically be added (fig. 4.12a) due to the ICP-style nature of this algorithm. This distributes the remaining error evenly over the rest of the trajectory, which is not constrained by loop closures (fig. 4.12b).

The optimization problem we now have to solve is to reduce the overall alignment error between all pairs of sweeps. This is an extension of the multiview error (3.4) error to continuous time and deformable registration. The registration becomes deformable, because we do not try to match rigid sweeps with each other, but match the surfels contained in each sweep against each other. These can move independently of each other which allows the sweep to deform during optimization. The extension to continuous time means that instead of estimating one absolute pose for a sweep, we estimate the trajectory during a sweep, represented as a weighted sum of a number of basis functions.

Note that the graph adjacency matrix of this continuous time deformable multiview error has a block structure of $N \times N$ for $N$ sweeps. Because of the symmetric surfel match error, we can save half the complexity and pick only the lower or upper diagonal part.

# 5 Evaluation

We evaluate our approach on synthetic data acquired with a specifically extended simulator that can simulate actuated 2D lidar scans and inertial measurements. The available ground truth allows to state the performance in different scenarios. In the following, we encode the different trajectory estimates or stages as follows:

| G | I | W | F |
|---|---|---|---|
| ground truth | initialization | windowed optimization | final global optimization |

## 5.1 Evaluation metrics

The difference between ground truth and estimated trajectory can be expressed using specific metrics developed for evaluation purposes [SEE+12]. Since these come from discrete-time approaches, we need to sample ground truth and estimated trajectory at corresponding times. Thus $\mathcal{Q} = \{Q_1, \cdots, Q_n\} \in SE(3)$ ground truth poses need to be compared to $\mathcal{P} = \{P_1, \cdots, P_n\} \in SE(3)$ pose estimates. The evaluation metrics are defined on corresponding pairs of poses and then summarized in a single number using a specific statistic.

The Relative Pose Error (RPE) at time step i with interval width 1 measures relative pose differences of consecutive poses. It thus describes the **local accuracy** of the trajectory, useful for evaluating drift.

$$E_i := (Q_i^{-1} Q_{i+1})^{-1} (P_i^{-1} P_{i+1})^{-1} \tag{5.1}$$
RPE

The Absolute Trajectory Error (ATE) at time step i measures absolute pose differences.[1] It thus describes the **global consistency** of the estimated trajectory, which is especially relevant after loop closures and global optimization.

$$F_i := Q_i^{-1} S P_i \tag{5.2}$$
ATE

Root Mean Squared (RMS) is a statistic to describe a set of scalar values compactly. Scalars are obtained by computing the norm of the translational components $\mathbf{t}_i$ of above metrics, because it encodes the rotational component [SEE+12]. Then they are summarized as $\sqrt{\frac{1}{n}\sum_{i=1}^{n} \|\mathbf{t}_i\|^2}$ to get a **single number for the whole trajectory**. In the following, we always mean their RMS when talking about the above evaluation metrics.

[1] Because both trajectories can live in different coordinate systems, we need the rigid body motion $S$ that aligns them in a least-squares sense. It is obtained by registering all the translational parts of $Q$ to the ones of $P$ using [AHB87], which is the usual registration method of point to point (3.1) ICP.

## 5.2 Scenarios

### 5.2.1 Scenario box

The first scenario (fig. 5.1) is a half-circular motion inside a box where the orientation changes continuously, aimed at testing our trajectory representation.

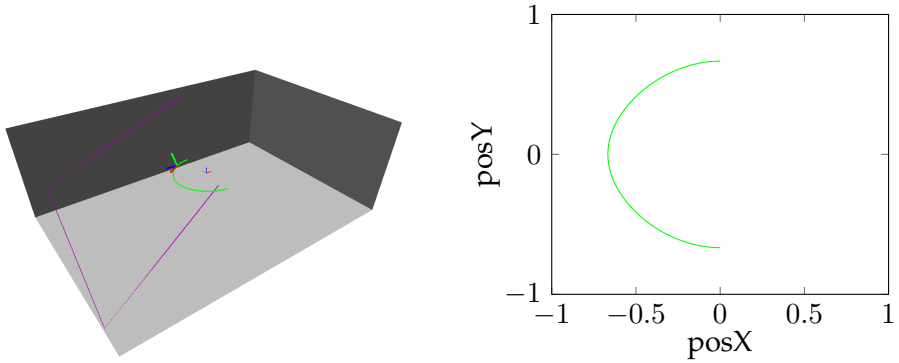



Figure 5.1: **box** A half-circular trajectory inside a box. The trajectory is drawn in green, on the right side there is a top down view of it. A simulated LiDAR scan is shown in magenta.

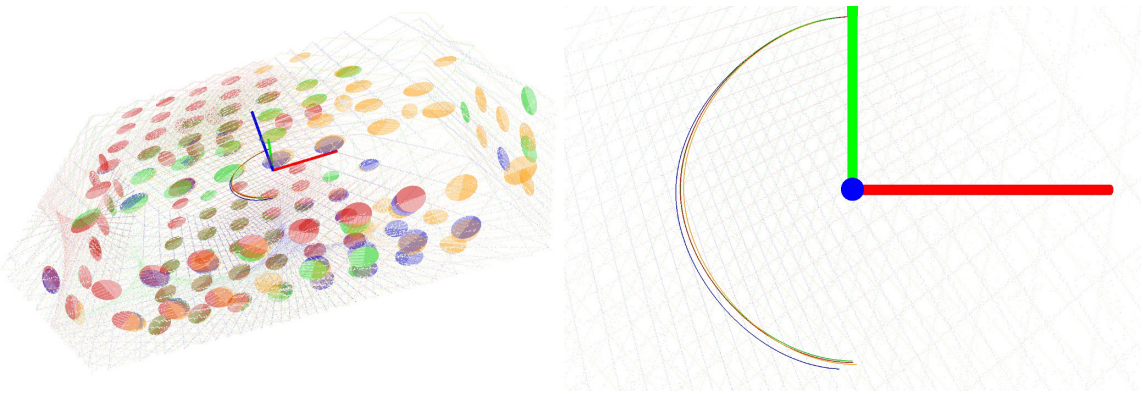


Figure 5.2: Left: The small circles are the covariance ellipses of the surfels, belonging to four different sweeps according to their color. Right: A top down view of estimated trajectories.

| stage | ate | rpe |
|---|---|---|
| I [mm] | 9.1 | 0.19 |
| ↓ [%] | 38 | 41 |
| W [mm] | 5.62 | 0.11 |
| ↓ [%] | 68 | 54 |
| F [mm] | 1.79 | $4.97 \cdot 10^{-2}$ |

| sweep | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 45 | 32 | 34 |
| 1 | 0 | 0 | 34 | 38 |
| 2 | 0 | 0 | 0 | 42 |
| 3 | 0 | 0 | 0 | 0 |

Table 5.1: Left: Evaluation metrics and their relative decrease during the different stages of the algorithm. Right: The initial adjacency matrix used in the final global optimization. Each entry depicts the number of surfel matches between the two sweeps.

This scenario consists of four sweeps (fig. 5.2). Our trajectory representation can closely follow even slight angular velocity changes during the circular motion, which allows to

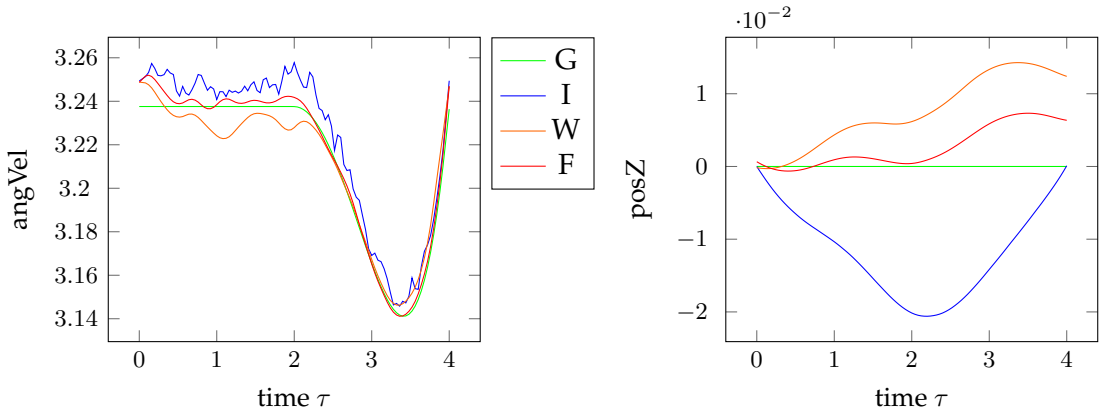



Figure 5.3: Left: Even with very noisy angular velocity measurements seen in the initial trajectory, the final trajectory follows the ground truth closely. Right: The z drift of the trajectory is minimized after each stage.

minimize the drift (fig. 5.3). The evaluation metrics decrease in each stage, which shows that the algorithm works as expected. The percentual decrease of the metrics during global optimization is significantly higher than between the first two stages (table 5.1). This is due to the fact that the graph adjacency matrix is fully connected, with about the equal amount of surfel matches in between the sweeps. These surfel matches allow to compensate very noisy angular velocity measurements.

### 5.2.2 Scenario loop

The second scenario (fig. 5.4) starts and ends in the same point, testing the ability to find loop closures. The scene has the topology of a donut or box with a whole in the middle, limiting the visibility of equal geometry during consecutive sweeps.

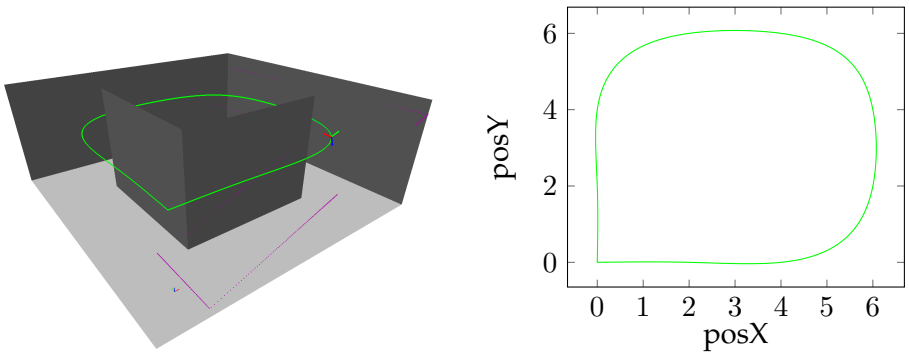



Figure 5.4: **loop** A trajectory that ends in its starting point. A top down view is shown as well. The scene is a box with a whole in the middle.

This scenario consists of 12 sweeps and the trajectory ends in the same position as it starts, only the orientations of start and end are 90 degrees apart (fig. 5.5). The algorithm is able to overcome large z drift by closing the loop (fig. 5.6). The adjacency matrix of the graph used in global optimization (table 5.2) is approximately band-diagonal, meaning that four consecutive sweeps can be matched against each other due to the visibility

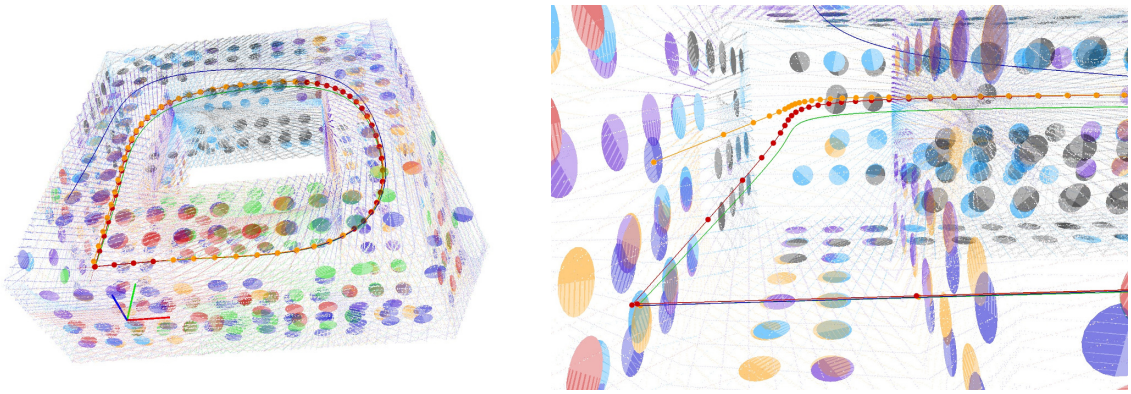


Figure 5.5: Left: The small circles are the covariance ellipses of the surfels, belonging to 12 different sweeps according to their color. The initial trajectory is off upwards. Right: The effect of the loop closure is shown in detail. The final (red) trajectory nearly ends at the start position again.

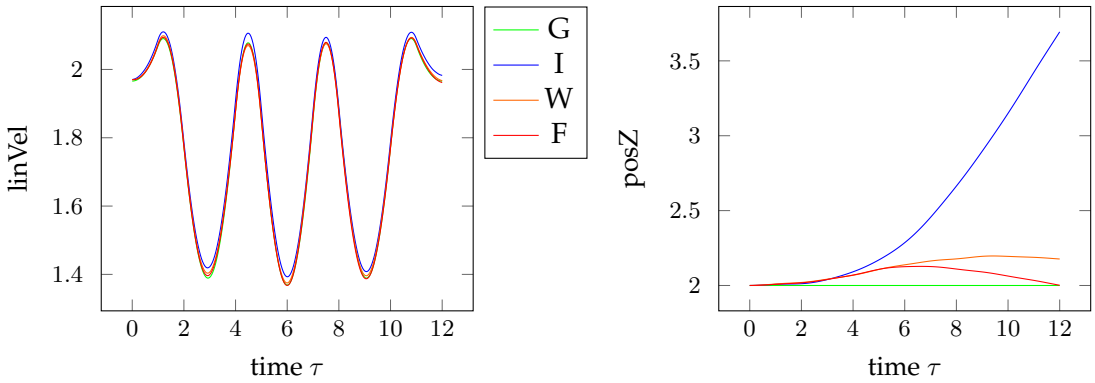



Figure 5.6: Left: The initial trajectory overestimates the true linear velocity, especially in z direction. Right: This causes the huge drift in the initial trajectory estimate. The open-loop solution is also affected. But after loop closure and global optimization, the z drift at the end is corrected.

| stage | ate | rpe |
|---|---|---|
| I [mm] | 327.06 | 1.57 |
| ↓ [%] | 86 | 88 |
| W [mm] | 43.98 | 0.17 |
| ↓ [%] | 87 | 53 |
| F [mm] | 5.42 | $8.05 \cdot 10^{-2}$ |

| sweep | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 38 | 13 | 10 | 2 | 0 | 0 | 0 | 6 | 8 | 11 | 41 |
| 1 | 0 | 0 | 32 | 17 | 4 | 1 | 0 | 0 | 0 | 2 | 5 | 32 |
| 2 | 0 | 0 | 0 | 30 | 16 | 7 | 2 | 1 | 2 | 0 | 0 | 14 |
| 3 | 0 | 0 | 0 | 0 | 41 | 22 | 10 | 4 | 2 | 0 | 0 | 9 |
| 4 | 0 | 0 | 0 | 0 | 0 | 40 | 14 | 7 | 2 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 31 | 23 | 8 | 6 | 0 | 2 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 41 | 23 | 10 | 2 | 1 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 31 | 14 | 3 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 34 | 20 | 9 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 42 | 18 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 27 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 5.2: Left: Evaluation metrics and their relative decrease during the different stages of the algorithm. Right: The initial adjacency matrix used in the final global optimization. Each entry depicts the number of surfel matches between the two sweeps.

constraints when turning around the 3 corners. Additionally, the loop-closure is visible in the upper right part of the matrix: the first three sweeps can be matched to the last three sweeps. The decrease of the evaluation metrics between the stages is nearly same for ATE (86% and 87%), but for RPE, it is smaller for global optimization (53%) than before (88%).

This is due to the fact that this large loop closure improves global consistency, but does not in the same manner reduce the drift, especially in the middle part of the trajectory.

### 5.2.3 Scenario stairs

The last scenario (fig. 5.7) contains fine-scale geometry features like stairs and vertical notches in the walls, testing the algorithm's 3D scene representation. The trajectory is the only one with a varying z coordinate, testing the ability to perform full 3D SLAM.

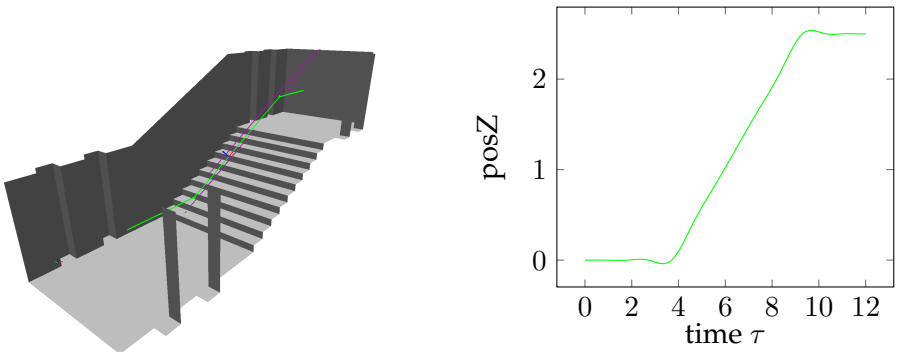



Figure 5.7: **stairs** A trajectory where y is 0, x grows linearly and z is plotted above. The scene contains fine-scale geometry features.

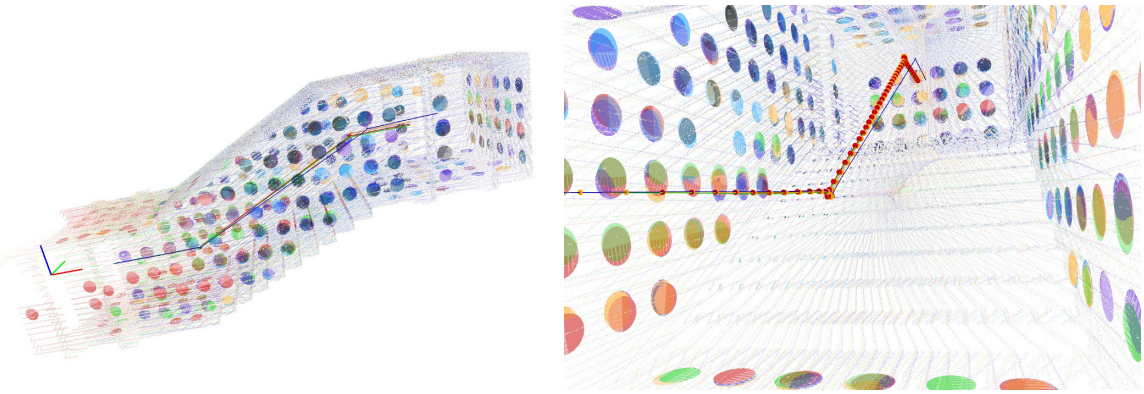


Figure 5.8: Left: Side view of the scenario. Right: Detail looking towards positive x. Note the y drift of the initial trajectory and that there are no surfels on the stairs.

The stairs scenario is the only one where the z coordinate is modified and which has fine-scale geometry features (fig. 5.8). No surfels are generated on the steps because the chosen voxel size is bigger than the steps. Thus, the threshold on the plane-likeness (4.10) discarded these unreliable surfels. Nevertheless, the algorithm still performs well and can minimize the drift in z direction (fig. 5.9). The adjacency matrix (table 5.3) is quite dense. About six consecutive frames have more than 10 surfel matches. This acts like fixed view or keyframe optimization, reducing the accumulated drift from the frame-to-frame time-windowed optimization. The relative decrease of the evaluation metrics (87%, 88%) and (83%, 44%) is very similar to the values of the previous loop closure scenario. This highlights the fact that our algorithm handles loop closure and keyframe optimization naturally in a very similar manner.

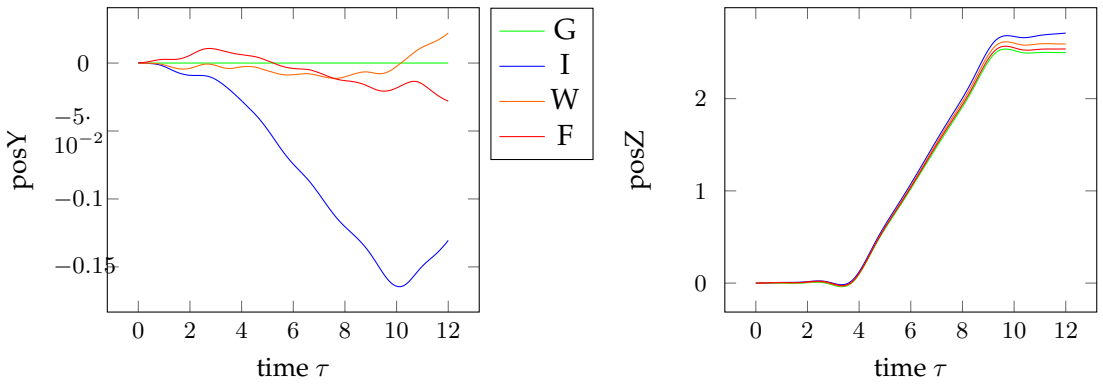



Figure 5.9: Left: Windowed and final trajectory estimate have about the same absolute drift in y direction. Right: Final trajectory nearly coincides with the solution in z direction.

| stage | ate | rpe |
|---|---|---|
| I [mm] | 229.04 | 0.75 |
| ↓ [%] | 87 | 83 |
| W [mm] | 28.42 | 0.13 |
| ↓ [%] | 88 | 44 |
| F [mm] | 3.38 | $7.05 \cdot 10^{-2}$ |

| sweep | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 54 | 23 | 38 | 37 | 27 | 20 | 11 | 8 | 6 | 6 | 5 |
| 1 | 0 | 0 | 25 | 35 | 41 | 38 | 31 | 17 | 12 | 10 | 8 | 8 |
| 2 | 0 | 0 | 0 | 17 | 18 | 16 | 13 | 9 | 6 | 3 | 2 | 2 |
| 3 | 0 | 0 | 0 | 0 | 52 | 49 | 37 | 31 | 25 | 17 | 17 | 17 |
| 4 | 0 | 0 | 0 | 0 | 0 | 98 | 83 | 68 | 57 | 39 | 28 | 27 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 109 | 98 | 78 | 51 | 44 | 36 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 99 | 88 | 59 | 48 | 44 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 91 | 60 | 53 | 46 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 59 | 53 | 49 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 44 | 46 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 54 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 5.3: The error and their relative decrease during the different stages of the algorithm. The initial adjacency matrix for global optimization. Each number depicts the number of surfel matches between the two sweeps.

## 5.3 Effect of noise

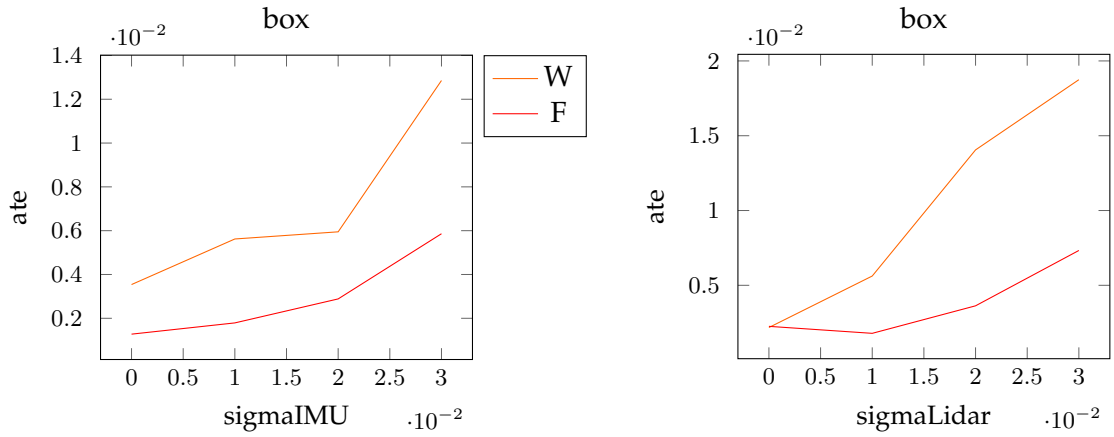



Figure 5.10: Effect of noise on IMU (left) and LiDAR measurements (right) on the absolute trajectory error for the box dataset. The different lines depict the trajectory estimates of the different stages of the algorithm.

Here we evaluate the effect of noise on our sensor measurements and its impact on accuracy. The noise model used for the IMU is white noise on angular velocity and linear

acceleration vector. Additionally, we add a constant bias with a value of the standard deviation of the white noise. The LiDAR data has white range noise along the laser beam. As expected, the accuracy improves when lowering the noise level (fig. 5.10). This general trend is clearly visible regardless of the sensor.

## 5.4 Effect of different parameters

Certain parameters of our algorithm affect the overall performance. First of all the number of ICP iterations (iter), the knot spacing of our B-spline (taus) and the size of our voxels (voxelSize) used for space discretization.

### 5.4.1 Number of ICP iterations (iter)

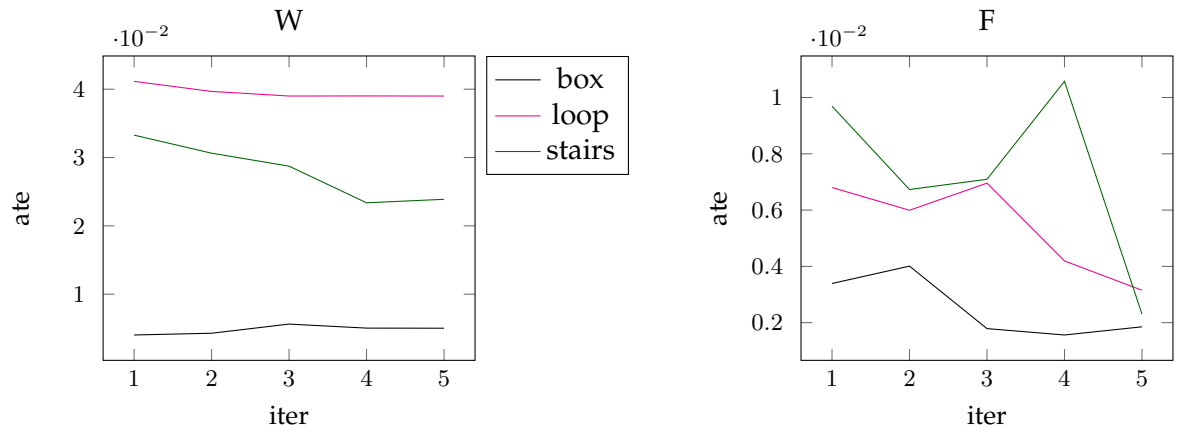



Figure 5.11: Effect of increasing the number of iterations (iter) for the time-windowed (left) and the final trajectory estimate (right). The different lines correspond to the scenarios.

Increasing the number of iterations mostly decreases the error, especially for the final trajectory estimate. If new but wrong correspondences are found after an ICP update step, it is possible that the error increases, like in the time-windowed estimate of the box trajectory (fig. 5.11) or the final estimate of the stairs trajectory when doing only four iterations. If no new correspondences are found, we are in a local minima, thus further iterations won't reduce the error anymore as in the time-windowed estimate of the box and loop trajectory at three and more iterations.

### 5.4.2 Knot spacing (taus)

The knot spacing of our B-spline trajectory representation has a dramatic effect on the performance. If it is too high, we cannot adequately represent high-frequency components of the trajectory. However, if it is too low, the optimization problem might become underconstrained if we do not have enough surfel matches between certain segments. The state size to optimize doubles when halving the knot spacing, and thus the computational complexity increases. The optimal value depends on the complexity of the motion to represent.

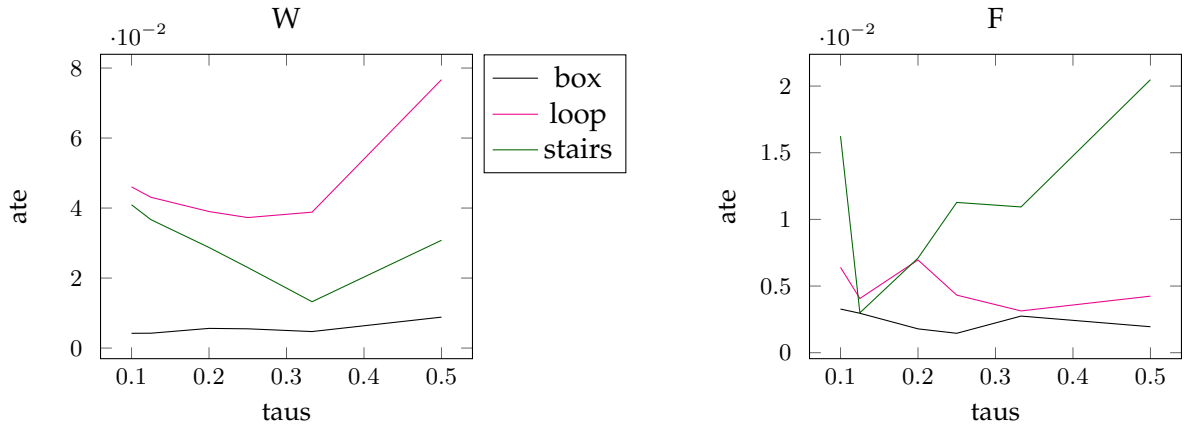



Figure 5.12: Effect of modifying the knot spacing (taus) $\tau_s$ for the time-windowed (left) and the final trajectory estimate (right). The different lines correspond to the scenarios.

It is thus advantageous to employ a rotating laser scanner with constant angular velocity instead of a nodding laser scanner where the angular velocity changes as a periodic function.

### 5.4.3 Space discretization (voxelSize)

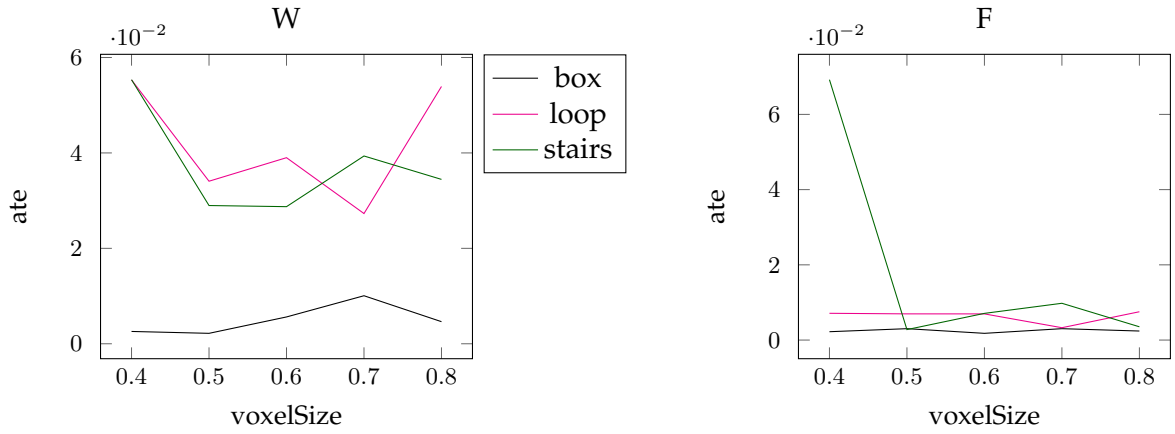



Figure 5.13: Effect of modifying the space discretization (voxelSize) for the time-windowed (left) and the final trajectory estimate (right). The different lines correspond to the scenarios.

The voxel size should be set accordingly to the scale at which most planar regions are expected to be encountered in the environment. Smaller voxels do not automatically reduce the error. This is especially visible in the stairs scenario. The size of the steps must be around 40cm. In fact, there is a correlation between the voxel size, the plane-likeness threshold, the magnitude of $\epsilon$ in the disc shape prior and the accuracy. These parameters together encode how geometry is processed and discretization noise accounted for. A complete treatment of the interplay however is beyond the scope of this work.

# 6 Summary

## 6.1 Conclusion

We developed a method that is able to estimate a continuous 3D trajectory from an actuated 2D laser scanner coupled to an IMU. Our method works especially well for indoor environments, where the planarity assumption holds. The properties of our chosen trajectory representation make it a perfect fit for continuous time trajectory estimation. The trajectory estimate is initialized using inertial navigation techniques on IMU measurements. The time-windowed processing is able to improve the initial trajectory estimate if there are enough surfel matches between consecutive sweeps. The global optimization step further allows to reduce the error by optimizing surfel matches and IMU constraints over the duration of the complete trajectory all at once. If initialized well enough, it additionally closes loops automatically, which further reduces the drift and distributes the remaining error evenly. Evaluation on synthetic data with different scenarios, noise levels and parameter settings showed the versatility, stability and adaptability of our algorithm. The experiments demonstrated the high accuracy of the algorithm, which is mostly limited by the computational power one wants to invest.

## 6.2 Future work

This work demonstrated the feasibility of our approach and focused on a thorough understanding of all the intrinsic implementation details. However, there are still lots of directions for extensions.

**Evaluation on real data and comparison to other approaches** This work focused on an intrinsic evaluation of the algorithm which is only possible with synthesized sensor data since it allows to test certain aspects and their accuracy compared to ground truth in isolation. With the confidence in basic stability under noise and knowledge about the adaptability options through different parameter settings, we could now test the algorithm on real data. An extrinsic evaluation of the approach comparing it to other methods would also be insightful. However, it seems that there is no publicly available benchmark for actuated 2D LiDAR. Instead of creating one on our own, it might be possible to adapt certain parts of our algorithm to other input modalities.

**Higher spline order** As suggested in the related work section, we should examine if a spline of even higher order can improve our trajectory's representational power even with a higher knot spacing and without introducing wiggles common to higher order interpolation methods. This would allow to further reduce the state size and thus accelerate the optimization.

**More robust loop closures** Loop closures are established during the correspondence step of ICP by matching local surface properties such as centroid and normal derived via PCA. This method is not very robust against false positive matches. It could be extended or complemented by a 2D or 3D feature-based loop-closure detection algorithm.

# Bibliography


[AB13] Sean Anderson and Timothy D Barfoot. Towards relative continuous-time slam. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1033–1040. IEEE, 2013.

[ABB14] Hatem Alismail, L Douglas Baker, and Brett Browning. Continuous trajectory estimation for 3d slam from actuated lidar. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6096–6101. IEEE, 2014.

[AHB87] K Somani Arun, Thomas S Huang, and Steven D Blostein. Least-squares fitting of two 3-d point sets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (5):698–700, 1987.

[AM] Sameer Agarwal and Keir Mierle. Ceres solver. http://ceres-solver.org.

[AMB15] Sean Anderson, Kirk MacTavish, and Timothy D Barfoot. Relative continuous-time slam. *The International Journal of Robotics Research*, pages 1453–1479, 2015.

[BM92] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992.

[BZ09] Michael Bosse and Robert Zlot. Continuous 3d scan-matching with a spinning 2d laser. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4312–4319. IEEE, 2009.

[BZF12] Michael Bosse, Robert Zlot, and Paul Flick. Zebedee: Design of a spring-mounted 3-d range sensor with application to mobile mapping. *IEEE Transactions on Robotics*, 28(5):1104–1119, 2012.

[CM91] Yang Chen and Gérard Medioni. Object modeling by registration of multiple range images. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 2724–2729. IEEE, 1991.

[DKL98] Erik B Dam, Martin Koch, and Martin Lillholm. *Quaternions, interpolation and animation*. Datalogisk Institut, Københavns Universitet, 1998.

[Far02] Gerald Farin. *Curves and Surfaces for CAGD*. 5 edition, 2002.

[FBS12] Paul Furgale, Timothy D Barfoot, and Gabe Sibley. Continuous-time batch estimation using temporal basis functions. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2088–2095. IEEE, 2012.

[FCDS15] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *Robotics: Science and Systems XI*, number 214687, 2015.

[FCF12] Simone Fantoni, Umberto Castellani, and Andrea Fusiello. Accurate and automatic alignment of range surfaces. In *3DIMPVT*, pages 73–80. Citeseer, 2012.

[Fit03] Andrew W Fitzgibbon. Robust registration of 2d and 3d point sets. *Image and Vision Computing*, 21(13):1145–1153, 2003.

[For14] Per-Erik 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.

[FTBS15] Paul Furgale, Chi Hay Tong, Timothy D Barfoot, and Gabe Sibley. Continuous-time batch trajectory estimation using temporal basis functions. *The International Journal of Robotics Research*, page 0278364915585860, 2015.

[GWA07] Mohinder S Grewal, Lawrence R Weill, and Angus P Andrews. *Global positioning systems, inertial navigation, and integration*. John Wiley & Sons, 2007.

[Haa15] Adrian Haarbach. 3d object reconstruction using point pair features. Bachelor's thesis, Technische Universität München, 2015.

[Han06] Andrew J. Hanson. *Visualizing Quaternions*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[Hol08] Darryl D Holm. *Geometric mechanics*. Imperial College Press, 2008.

[Jia08] Yan-Bin Jia. Quaternions and rotations. *Com S*, 477(577):15, 2008.

[KKS95] 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*, pages 369–376. ACM, 1995.

[KZB16] Lukas Kaul, Robert Zlot, and Michael Bosse. Continuous-time three-dimensional mapping for micro aerial vehicles with a passively actuated rotating laser scanner. *Journal of Field Robotics*, 33(1):103–132, 2016.

[Low04] Kok-Lim Low. Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, 2004.

[LPPS13] 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 *BMVC*, 2013.

[Mor97] Michael E. Mortenson. *Geometric Modeling (2nd Ed.)*. John Wiley & Sons, Inc., New York, NY, USA, 2 edition, 1997.

[MSKS05] Y. Ma, S. Soatto, J. Kosecká, and S.S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Interdisciplinary Applied Mathematics. Springer New York, 2005.

[Par12] Rick Parent. *Computer animation: algorithms and techniques*. Newnes, 3 edition, 2012.

[PGK02] Mark Pauly, Markus Gross, and Leif P Kobbelt. Efficient simplification of point-sampled surfaces. In *Proceedings of the conference on Visualization'02*, pages 163–170. IEEE Computer Society, 2002.

[PPLS15] Alonso Patron-Perez, Steven Lovegrove, and Gabe Sibley. A spline-based trajectory representation for sensor fusion and rolling shutter cameras. *International Journal of Computer Vision*, 113(3):208–219, 2015.

[RL01] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE, 2001.

[SEE+12] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012.

[SFSF15] 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.

[SG15] Jacopo Serafin and Giorgio Grisetti. Nicp: Dense normal based point cloud registration. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 742–749. IEEE, 2015.

[Sho85] Ken Shoemake. Animating rotation with quaternion curves. In *ACM SIGGRAPH computer graphics*, volume 19, pages 245–254. ACM, 1985.

[Sho87] Ken 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.

[SHT09] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics: Science and Systems*, volume 2, 2009.

[TW04] David Titterton and John L Weston. *Strapdown inertial navigation technology*, volume 17. IET, 2004.

[Woo07] Oliver J Woodman. An introduction to inertial navigation. *University of Cambridge, Computer Laboratory, Tech. Rep. UCAMCL-TR-696*, 14:15, 2007.

[WW91] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques*. ACM, New York, NY, USA, 1991.

[WW03] Oliver Wulf and Bernardo Wagner. Fast 3d scanning methods for laser measurement systems. In *International conference on control systems and computer science (CSCS14)*, pages 2–5. Citeseer, 2003.