

# Trajectory Representation and Landmark Projection for Continuous-Time Structure from Motion

Hannes Ovrén<sup>1</sup> and Per-Erik Forssén<sup>1</sup>

Journal Title  
XX(X):1–14  
©The Author(s) 2016  
Reprints and permission:  
[sagepub.co.uk/journalsPermissions.nav](http://sagepub.co.uk/journalsPermissions.nav)  
DOI: 10.1177/ToBeAssigned  
[www.sagepub.com/](http://www.sagepub.com/)  


## Abstract

This paper revisits the problem of continuous-time structure from motion, and introduces a number of extensions that improve convergence and efficiency. The formulation with a  $C^2$ -continuous spline for the trajectory naturally incorporates inertial measurements, as derivatives of the sought trajectory. We analyse the behaviour of split interpolation on  $\mathbb{SO}(3)$  and on  $\mathbb{R}^3$ , and a joint interpolation on  $\mathbb{SE}(3)$ , and show that the latter implicitly couples the direction of translation and rotation. Such an assumption can make good sense for a camera mounted on a robot arm, but not for hand-held or body-mounted cameras. Our experiments show that split interpolation on  $\mathbb{R}^3$  and  $\mathbb{SO}(3)$  is preferable over  $\mathbb{SE}(3)$  interpolation in all tested cases. Finally, we investigate the problem of landmark reprojection on rolling shutter cameras, and show that the tested reprojection methods give similar quality, while their computational load varies by a factor of 2.

## Keywords

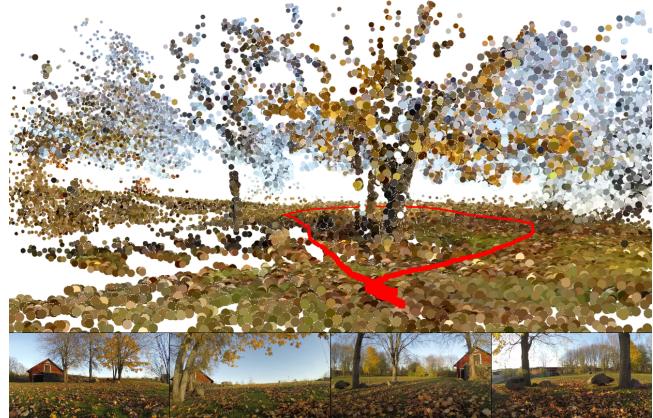
Sensor Fusion, Computer Vision, SLAM, Rolling Shutter, Structure from Motion

## 1 Introduction

Structure from motion on video, is a variant of the Simultaneous Localisation And Mapping (SLAM) problem, which by now is one of the classical problems in robotics (Bailey and Durrant-Whyte 2006). Structure from motion on video has a wide range of applications, such as 3D mapping (Engel et al. 2016), video stabilization (Kopf et al. 2014), and autonomous navigation (Bailey and Durrant-Whyte 2006). Traditionally such systems used discrete-time camera poses, while this paper considers the more recent continuous-time formulation (Furgale et al. 2015). Many SLAM systems exploit a combination of sensors for robustness; LIDAR, cameras, and inertial sensors (typically gyroscopes and accelerometers) are popular choices. It is well known that cameras and inertial sensors are complementary, and thus useful to combine. Primarily this is because inertial measurements have biases, that can be estimated during fusion with camera measurements. In addition, cameras often provide very accurate relative pose, but not absolute scale, and camera-only structure from motion fails in the absence of scene structure.

Platforms that house both cameras and inertial sensors are now very common. Examples include most current smartphones and tablets, but also some action cameras, e.g. newer models from GoPro. Nearly all such platforms use cameras with an electronic *rolling shutter* mechanism, that acquires each frame in a row-by-row fashion. This lends itself naturally to continuous-time motion models, as the camera has a slightly different pose in each image row.

Classical structure from motion treats camera trajectories as a set of discrete poses (Triggs et al. 2000), but by replacing the poses with spline knots, we obtain the continuous-time formulation, which is used on rolling shutter cameras for video structure from motion (Hedborg et al. 2012).



**Figure 1.** Rendered model estimated on the **RC-Car** dataset, using split interpolation. Top: model rendered using Meshlab. Bottom: Sample frames from dataset.

A useful property of the continuous pose representation, introduced by Furgale et al. (2012), is that its derivatives can predict measurements from an inertial measurement unit (IMU), which simplifies fusion of data from cameras and IMUs, and multi-sensor platforms in general (Furgale et al. 2015). Continuous-time structure from motion is also crucial in camera-IMU calibration when the camera has a rolling shutter (Ovrén and Forssén 2015; Furgale et al. 2015;

<sup>1</sup>Linköping University, Sweden

### Corresponding author:

Hannes Ovrén, Computer Vision Laboratory, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden  
Email: hannes.ovren@liu.se

Lovegrove et al. 2013). Compared to classical structure from motion, the continuous-time version has a moderate increase in complexity, due to reduced sparsity of the system Jacobian as shown by Hedborg et al. (2012).

### 1.1 Contributions

In this paper we revisit the continuous-time structure from motion problem with inertial measurements, and rethink several design choices:

- We replace the  $\text{SE}(3)$ -based interpolation used in the *Spline Fusion* method (Lovegrove et al. 2013; Patron-Perez et al. 2015) with a split interpolation in  $\mathbb{R}^3$  and  $\text{SO}(3)$ . This leads to a trajectory representation that does not couple rotation and translation in a screw motion, see Figure 6, and is better suited to e.g., hand-held camera motions.
- We compare the split and  $\text{SE}(3)$  trajectory representations theoretically, and in a series of both synthetic and real data experiments.
- We compare the performance and efficiency of three previously proposed ways to incorporate reprojection time into the optimization (Hedborg et al. 2012; Furgale et al. 2012; Lovegrove et al. 2013; Kim et al. 2016).
- For completeness, we also describe our recently published *spline error weighting* approach to better balance the residuals in the optimization problem, and to automatically set the spline knot spacing based on desired trajectory accuracy (Ovrén and Forssén 2018).

The main goal of the paper is to help other researchers make informed choices when designing their continuous-time structure from motion systems.

### 1.2 Related work

The classical pose interpolation approach in computer animation is to independently interpolate the camera orientation in the orientation group  $\text{SO}(3)$  and the camera positions in the vector space  $\mathbb{R}^3$  (Kim et al. 1995).

In robotic animation it is instead common to do direct interpolation on the special Euclidean group  $\text{SE}(3)$  (Crouch et al. 1999). Recently, such a direct interpolation on  $\text{SE}(3)$  was applied to the continuous-time structure from motion problem, by integrating the  $\text{SE}(3)$  spline into an optimization framework (Lovegrove et al. 2013; Patron-Perez et al. 2015). This formulation generalizes the orientation interpolation of Kim et al. (1995) to  $\text{SE}(3)$ . Several recent continuous-time structure from motion papers use the  $\text{SE}(3)$  approach (Patron-Perez et al. 2015; Kerl et al. 2015; Kim et al. 2016; Anderson and Barfoot 2015), while others use separate interpolation of pose and orientation (Furgale et al. 2015; Oth et al. 2013). In the following sections, we analyse the two approaches theoretically, and also compare them experimentally.

When re-projecting a landmark in a frame there is an additional complication in the rolling shutter case. As one image coordinate (typically the image row) corresponds to observation time, the reprojection of a landmark at time  $t$  will not necessarily end up at the row corresponding to that time. Early methods handled this by setting the reprojection time to

the landmark observation time (Hedborg et al. 2012; Furgale et al. 2012). This was improved upon by Oth et al. (2013) who also linearize the reprojection time error and convert it to a spatial error covariance. Lovegrove et al. (2013) instead use the Newton method to iteratively find a reprojection time with a consistent row coordinate, and this approach is also followed by Kerl et al. (2015). Yet another approach is to add a projection time parameter for each landmark observation, as well as a cost term for the projection time deviation (Kim et al. 2016). In the experiments, we refer to this approach as *lifting*, which is the common term for elimination of alternating optimization by adding variables and constraints (Zach 2014). No previous publication has compared these choices, instead each paper makes a hard commitment to one of the methods. In Furgale et al. (2015) some of the choices are discussed, but a comparison is left for future work.

### 1.3 Paper overview

The remainder of the paper is organized as follows. In section 2 we introduce the visual-inertial fusion problem that is the context of this paper. In section 3 we describe three methods for rolling shutter landmark projections, and in section 4 we present two different choices of continuous trajectory representation. Finally, in 5 we evaluate our methods experimentally, and section 6 summarizes the paper and gives an outlook.

Illustrations and plots are best viewed in colour.

## 2 Visual-inertial fusion

This work is an extension of the *Spline fusion* visual-inertial fusion framework introduced by Lovegrove et al. (2013). In this section we outline how the Spline fusion method works, and also summarize the improvements to robustness of the framework, introduced by Ovrén and Forssén (2018).

### 2.1 Video structure from motion

In structure from motion, the goal is to estimate camera poses, and 3D structure, from a set of images. If the images are from video, or are taken in sequence, the camera poses can be thought of as a trajectory over time.

A camera pose consists of a rotational component  $\mathbf{R} \in \text{SO}(3)$ , and a translational component  $\mathbf{p} \in \mathbb{R}^3$ . In standard structure from motion, the camera path is simply the set of all camera poses, with one pose per image,  $n$ :

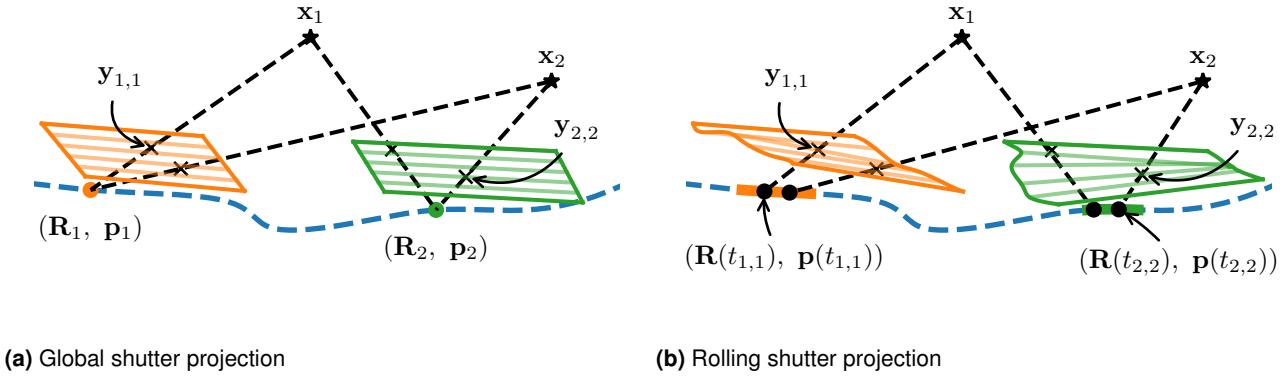
$$\mathbf{T}_n = (\mathbf{R}_n, \mathbf{p}_n). \quad (1)$$

We follow the convention in Patron-Perez et al. (2015), and define the pose such that  $\mathbf{T}$  is a transformation from the body (i.e. camera) to the global coordinate frame.

The objective is then to find the camera poses, and 3D points that minimize the cost function

$$J(\mathcal{T}, \mathcal{X}) = \sum_{\mathbf{T}_n \in \mathcal{T}} \sum_{\mathbf{x}_k \in \mathcal{X}} \|\mathbf{y}_{k,n} - \pi(\mathbf{T}_n^{-1} \mathbf{x}_k)\|^2. \quad (2)$$

Here,  $\mathcal{X}$  is the set of all 3D points,  $\mathcal{T}$  is the set of all camera poses, and  $\mathbf{y}_{k,n}$  is the observation of 3D point  $k$  in image  $n$ . The function  $\pi(\cdot)$  projects a 3D point in the camera coordinate frame to the image plane, using some



**Figure 2.** Structure from motion under global and rolling shutter geometry. Here,  $\mathbf{x}_k$  is a 3D landmark which is projected to an image observation,  $\mathbf{y}_{k,n}$ , in image  $n$ . Cameras are represented by their image plane, where we also show a limited number of the image rows. On the camera trajectory (dashed, blue line) we indicate the time instance (global shutter), or time span (rolling shutter), when the image was captured.

camera model. This formulation of the structure from motion objective is called *bundle adjustment* (Triggs et al. 2000). We illustrate the structure from motion geometry in Figure 2a.

## 2.2 Rolling shutter

In the previous section, we assumed that there is one camera pose per image, such that all pixels are captured at the same time. Such cameras are said to have a *global shutter*.

Most cameras available today are however equipped with a *rolling shutter* (Gamal and Eltoukhy 2005). Here, the image is read out from the sensor one row at a time, i.e. different rows are captured at different times. If the camera is moving while the image is captured, then we no longer have a single camera pose per image, but instead one camera pose per row. We illustrate the rolling shutter geometry in Figure 2b, where the camera pose at the row that corresponds to image observation  $\mathbf{y}_{k,n}$  is denoted  $(\mathbf{R}(t_{k,n}), \mathbf{p}(t_{k,n}))$ .

It has been shown (Hedborg et al. 2012) that ignoring rolling shutter when minimizing (2) reduces accuracy, and can even lead to reconstruction failures.

## 2.3 Continuous-time structure from motion

To handle the rolling shutter problem, the standard, or discrete-time, formulation of structure from motion in (1) can be modified to instead model the camera trajectory as a continuous-time function

$$\mathbf{T}(t) = (\mathbf{R}(t), \mathbf{p}(t)). \quad (3)$$

Instead of being restricted to a set of discrete camera poses, we can now determine the camera pose at any time instance  $t$ . There are many ways to construct  $\mathbf{T}(t)$ , but arguably the most common approach is to model it as some kind of spline.

Given this new representation, we modify the cost function (2) to

$$J(\mathcal{T}, \mathcal{X}) = \sum_{\mathbf{y}_{k,n} \in \mathcal{Y}} \|\mathbf{y}_{k,n} - \pi(\mathbf{T}^{-1}(t_{k,n})\mathbf{x}_k)\|^2. \quad (4)$$

where  $\mathcal{Y}$  is the set of all image observations, and  $\mathcal{X}$  is still the set of 3D points. However,  $\mathcal{T}$  is no longer a set of discrete

camera poses, but is instead the set of *trajectory parameters*. The exact nature of the trajectory parameters depends on how we choose to model the trajectory.

With a continuous-time formulation, structure from motion can be solved on both rolling shutter, and global shutter cameras by minimizing the same cost function (4). There are however some practical aspects regarding how the landmarks are projected into the camera, which we will further investigate in section 3.

Next, we will show another new possibility: incorporating inertial measurements in the bundle adjustment formulation.

## 2.4 Inertial measurements

An IMU consists of a gyroscope, which measures angular velocities,  $\boldsymbol{\omega}$ , and an accelerometer, which measures linear accelerations,  $\mathbf{a}$ . These measurements are direct observations of motion, and are a useful addition to the trajectory estimation problem. Lovegrove et al. (2013) therefore extend (4) to also include residuals for the gyroscope and accelerometer measurements:

$$\begin{aligned} J(\mathcal{T}, \mathcal{X}) = & \sum_{\mathbf{y}_{k,n} \in \mathcal{Y}} \|\mathbf{y}_{k,n} - \pi(\mathbf{T}^{-1}(t_{k,n})\mathbf{x}_k)\|^2 \\ & + \sum_m \|\boldsymbol{\omega}_m - \nabla_\omega \mathbf{T}(t_m)\|_{\mathbf{W}_g}^2 \\ & + \sum_l \|\mathbf{a}_l - \nabla_a^2 \mathbf{T}(t_l)\|_{\mathbf{W}_a}^2. \end{aligned} \quad (5)$$

The operators  $\nabla_\omega$  and  $\nabla_a^2$  represent inertial sensor models which predict gyroscope and accelerometer values given the trajectory model  $\mathbf{T}(t)$ , using analytic differentiation. The norm weight matrices  $\mathbf{W}_g$  and  $\mathbf{W}_a$  are used to balance the three modalities fairly. We show how to set the norm weight matrices in section 2.6.1.

For best results, the inertial sensor models,  $\nabla_\omega$  and  $\nabla_a^2$ , should model the used sensors as well as possible. At the very least they should account for a constant measurement bias, however, more advanced models that include e.g. axis misalignment, or time-varying biases, are also possible. In section 4 we derive basic inertial sensor models for the trajectories which we are interested in.

Looking at the IMU residuals in (5), we can see that there are two things that makes it problematic to use a discrete camera trajectory here. Firstly, the IMU measurement timestamps do not in general coincide with the frame times. This is partly because the IMU is usually sampling at a much higher rate than the camera. With a trajectory consisting only of discrete poses, it is not obvious how to extract a pose for these intermediate timestamps. The continuous-time formulation does not have this problem, since it allows us to determine the camera pose at any given time instance.

Secondly, the IMU residuals require us to compute derivatives of the trajectory, to get angular velocity and linear acceleration, respectively. With a discrete-time trajectory, these derivatives are not available. A continuous-time trajectory can, however, be built such that the required derivatives exist. To avoid derivatives, discrete time systems commonly use sensor integration instead. However, whenever the sensor bias is updated, the sensor integration has to be recomputed. Much effort has thus been spent to improve performance of sensor integration in the discrete pose case (Forster et al. 2015).

Since we need second order derivatives to compute the acceleration, it is crucial that the trajectory representation  $\mathbf{T}(t)$  is  $C^2$ -continuous. This is the reason why cubic B-splines are a popular choice.

## 2.5 Splined trajectories

Splines are an excellent choice for representing a continuous trajectory because their derivatives can be easily computed analytically. To introduce the general concept of splines, we will first describe it in only one dimension. In section 4 we then describe how splines can be used to model a continuous camera pose.

A spline consists of a set of *control points*,  $\Theta = (\theta_1, \dots, \theta_K)$ , which are positioned at *knots*,  $(t_1, \dots, t_K)$ , in time. The value of the spline at a specific time  $t$  is computed from the control points, which are weighted by a *basis function*,  $B(t)$ . If the knots are evenly spaced,  $\Delta t$  apart, we have  $t_k = k\Delta t$ , and say that the spline is *uniform*:

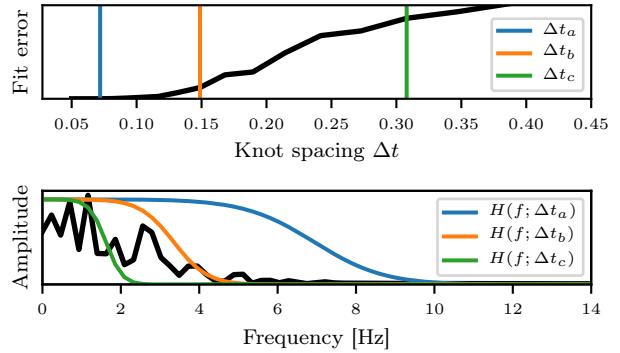
$$f(t|\Theta) = \sum_{k=1}^K \theta_k B(t - k\Delta t). \quad (6)$$

Fitting data to a (uniform) spline means to optimize the spline control points,  $\Theta$ , such that the shape of the spline matches the measurements. The knot spacing,  $\Delta t$ , is a hyper parameter, and in section 2.6.2 we show one way to set it to an appropriate value.

## 2.6 Spline Error Weighting

Before we attempt to minimize (5), using a splined trajectory, there are three hyper-parameters that must be set to appropriate values: the knot spacing,  $\Delta t$ , and the IMU residual norm matrices,  $\mathbf{W}_a$  and  $\mathbf{W}_g$ .

Lovegrove et al. (2013), who introduced (5), used a fixed knot spacing value of  $\Delta t = 0.1$ , and set the norm weight matrices to the inverse covariance of the respective measurement noises. In Ovrén and Forssén (2018) we showed why these choices are suboptimal, and derived



**Figure 3.** Top: Interpolation error for a test signal  $x(t)$  as a function of spline knot spacing  $\Delta t$ . Bottom: The frequency spectrum of  $x(t)$  (black) together with the frequency function  $H(f; \Delta t)$  for different choices of  $\Delta t$ .

a robust method to set these values. We will now give a summary of this method, which is called *Spline Error Weighting*.

**2.6.1 Selecting the IMU weights.** If we use inverse covariances to weight the inertial measurements in (5), then we make the implicit assumption that the chosen trajectory parameterization can perfectly represent the real motion.

However, a high  $\Delta t$  (sparse spline) results in a smooth trajectory which might not be able to fully represent the real motion. In this case the residuals in (5) will consist of two error terms: the measurement noise, and a *spline approximation error*. The Spline fusion method only accounts for the former, and in Ovrén and Forssén (2018) we showed that ignoring the approximation error leads to reconstruction failures.

Spline fitting can be characterized in terms of a frequency response function,  $H(f)$ , see Unser et al. (1993). In this formulation, a signal  $x(t)$  with the Discrete Fourier Transform (DFT)  $X(f)$  will have the frequency content  $(H \cdot X)(f)$  after spline fitting. In Figure 3 we show examples of how the spline interpolation function  $H(f)$ , and the spline fit error, depends on the choice of knot spacing.

By denoting the DFT of the frequency response function by the vector  $\mathbf{H}$ , and the DFT of the signal by  $\mathbf{X}$ , we can express the error introduced by the spline fit as:

$$\mathbf{E} = (1 - \mathbf{H}) \cdot \mathbf{X}. \quad (7)$$

This results in an approximation error variance

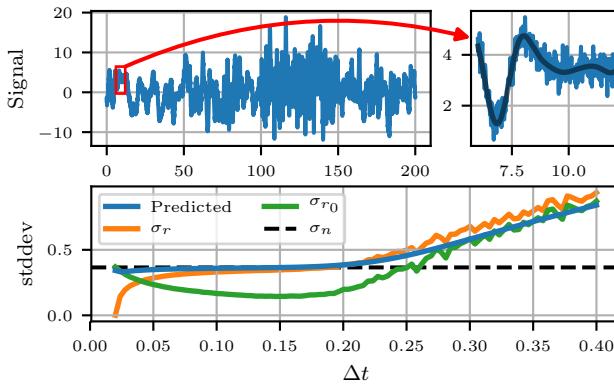
$$\hat{\sigma}_e^2 = \|\mathbf{E}\|^2/N = \|(1 - \mathbf{H}) \cdot \mathbf{X}\|^2/N, \quad (8)$$

where  $N$  is the number of samples. The residual weight matrices in (5) are then computed as

$$\mathbf{W}_r = \frac{1}{\hat{\sigma}_r^2} \mathbf{I} \text{ where } \hat{\sigma}_r^2 = \hat{\sigma}_e^2 + \hat{\sigma}_f^2. \quad (9)$$

Here  $\hat{\sigma}_f^2$  is a filtered version of the sensor noise variance  $\sigma_n^2$ , to account for the fact that  $\mathbf{X}$  used in (7) already contains this noise.

In Figure 4 we illustrate a simple experiment that demonstrates the behaviour of the Spline Error Weighting



**Figure 4.** Top: test signal and noise (right subplot is a detail). Bottom: standard deviations as functions of knot spacing.  $\sigma_r$  is the empirical residual standard deviation,  $\sigma_n$  is the noise standard deviation, which is used in Patron-Perez et al. (2015) to predict  $\sigma_r$ . *Predicted* is the Spline Error Weighting residual noise prediction.  $\sigma_{r_0}$  is the residual with respect to the noise-free signal  $x_0(t)$ .

residual error prediction (9). In Figure 4 top left, we show a test signal,  $x(t)$ , which is the sum of a true signal,  $x_0(t)$ , and white Gaussian noise  $n(t)$  with variance  $\sigma_n^2$ . The true signal has been generated by filtering white noise to produce a range of different frequencies and amplitudes. In figure 4 top right, we show a detail of the signal, where the added noise is visible.

We now apply a least-squares spline fit to the signal  $x(t)$ , to obtain the spline  $\hat{x}(t)$ , defined as in (6). This is repeated for a range of knot spacings,  $\Delta t$ , each resulting in a different residual  $r(t) = x(t) - \hat{x}(t)$ . The residual standard deviation  $\sigma_r$  is plotted in Figure 4, bottom. We make the same plot for the residual  $r_0(t) = x_0(t) - \hat{x}(t)$  which measures the error compared to the true signal. The resulting  $\sigma_{r_0}$  curve has a minimum at approximately  $\Delta t = 0.15$ , which is thus the optimal knot spacing. The fact that the actual residual  $\sigma_r$  decreases for knot spacings below this value thus indicates overfitting. From this short experiment, we can also see that the implicit assumption made in Patron-Perez et al. (2015) that the noise standard deviation  $\sigma_n$  can predict  $\sigma_r$  is reasonable for knot spacings at or below the optimal value. However, for larger knot spacings (at the right side of the plot) this assumption becomes increasingly inaccurate.

**2.6.2 Selecting the knot spacing.** In Patron-Perez et al. (2015) the spline knot spacing is fixed to  $\Delta t = 0.1$ . However, instead of deciding on a knot spacing explicitly, a more convenient design criterion is the amount of approximation error introduced by the spline fit. To select a suitable knot spacing,  $\Delta t$ , we thus first decide on a *quality value*,  $\hat{q} \in (0, 1]$ , that corresponds to the fraction of signal energy we want the approximation to retain. For a given signal,  $x(t)$ , with the DFT,  $\mathbf{X}$ , we define the quality value as the ratio between the energy, before and after spline fitting:

$$q(\Delta t) = \frac{\|\mathbf{H}(\Delta t) \cdot \mathbf{X}\|^2}{\|\mathbf{X}\|^2} \quad (10)$$

To find a suitable knot spacing for the signal, we search for the largest knot spacing  $\Delta t$  for which  $q(\Delta t) \geq \hat{q}$ .

The signals  $\mathbf{X}$  are based on the accelerometer, and gyroscope measurements, since these contain information

about both orientation and translation. See Ovrén and Forssén (2018) for further details.

**2.6.3 Adding a robust error norm.** In Patron-Perez et al. (2015), the cost function is defined as in (5), which assumes that the measurements are drawn from a zero-mean (Gaussian) distribution. This is a useful model for the IMU measurements, if we account for the sensor biases, but not for the image measurements. The image measurements are produced by tracking or feature matching over a sequence of images. The associations made are not perfect, and the risk of producing a feature track where the measurements do not correspond to one single 3D point is significant. Depending on the environment, we might also have moving objects in the scene, which can be successfully tracked, but are obviously not good landmarks.

Since such *outliers* do not correspond to the geometry we are trying to estimate, their errors can easily be orders of magnitude larger than those of the inlier set. If the outliers are not removed, the least-squares solver will try to bring these large errors down, even if it means that all the other measurement residuals (those in the inlier set) are increased. In standard structure from motion with global shutter cameras, most outliers can be removed by enforcing geometric consistency between observed image points. For rolling shutter cameras, enforcing geometric consistency is much harder, because the images no longer have a single corresponding camera pose. We instead accept that we will have at least some outliers, and try to mitigate their effect. We do this by introducing a *robust error norm* (Zhang 1997) which scales the residuals such that large residuals have less impact. The cost function is thus modified to its final formulation

$$\begin{aligned} J(\mathcal{T}, \mathcal{X}) = & \sum_{\mathbf{y}_{k,n} \in \mathcal{Y}} \phi(\mathbf{y}_{k,n} - \pi(\mathbf{T}^{-1}(t_{k,n}) \mathbf{x}_k)) \\ & + \sum_n \|\boldsymbol{\omega}_n - \nabla_{\omega} \mathbf{T}(t_n)\|_{\mathbf{W}_g}^2 \\ & + \sum_l \|\mathbf{a}_l - \nabla_a^2 \mathbf{T}(t_l)\|_{\mathbf{W}_a}^2, \end{aligned} \quad (11)$$

where  $\phi(x)$  is a robust error norm. In Ovrén and Forssén (2018), as well as in this work,  $\phi(x)$  is the Huber norm.

### 3 Rolling shutter projection

In (2) and (4) the landmark projection function  $\pi(\cdot)$  was defined to simply project a 3D point to its image plane location. This formulation works fine in the case of a global shutter camera, where there is a single camera pose for each captured image. In a rolling shutter camera, the image rows are captured and read out sequentially, which results in each row having its own camera pose. This means that an image observation

$$\mathbf{y}_{k,n} = [u, v]^T = \pi(\mathbf{T}^{-1}(t_{k,n}) \mathbf{x}_k) \quad (12)$$

was captured at time

$$t_{k,n} = t_n^0 + r \frac{v}{N_v}. \quad (13)$$

Here  $t_n^0$  is the time of the first row of frame  $n$ ,  $N_v$  is the number of image rows, and  $r$  is the rolling shutter *image*

*readout time.*  $r$  is simply the time it takes to read out a frame from the camera sensor.

The astute reader may have noticed a problem with equations (12) and (13): the projection time  $t_{k,n}$  requires knowledge of the projection row,  $v$ , but at the same time, the projection row also depends on the projection time! One of the contributions of this work are to analyze different methods for solving this chicken and egg problem. Before doing that, we will however have to replace the landmark projection function  $\pi(\cdot)$ .

### 3.1 The rolling shutter transfer function, $\psi$

So far we have represented a landmark  $k$  as a 3D point  $\mathbf{x}_k \in \mathbb{R}^3$ . This is, however, not the only possible parameterization. In Patron-Perez et al. (2015), whose approach we follow, a landmark is instead represented by its first observation  $\mathbf{y}_{k,*}$  and a corresponding *inverse depth*,  $\rho_k$ .

The inverse depth formulation has the nice property that it is easy to represent points at infinity by setting  $\rho_k = 0$ . It also means that the number of landmark parameters shrinks from  $3N$  to  $N$ , because only  $\rho_k$  has to be optimized for instead of the full 3D point  $\mathbf{x}_k$ .

With the inverse depth landmark representation we redefine the image measurement process to instead use a *rolling shutter transfer function*,  $\psi(\cdot)$ :

$$\begin{aligned}\mathbf{y}_{k,n} &= \psi(\mathbf{y}_{k,*}, \mathbf{T}^{-1}(t_{k,n})\mathbf{T}(t_{k,*}), \rho_k) \\ &= \pi\left(\mathbf{T}^{-1}(t_{k,n})\mathbf{T}(t_{k,*}) \begin{bmatrix} \pi^{-1}(\mathbf{y}_{k,*}) \\ \rho_k \end{bmatrix}\right).\end{aligned}\quad (14)$$

$\psi(\cdot)$  is called a transfer function because it transfers the reference observation  $\mathbf{y}_{k,*}$ , at time  $t_{k,*}$ , to a new measurement at image  $n$ , using the inverse depth,  $\rho_k$ , and the trajectory  $\mathbf{T}(t)$ .

For brevity, we will mostly use the shorter form  $\psi(t)$ , which should be understood as the projection (reference observation transfer) at time  $t$  for some landmark and trajectory.

In the following sections we describe three different strategies to implement  $\psi(\cdot)$ . One important property of each method, is how well they handle the *rolling shutter time deviation*

$$\epsilon(t_{k,n}) = (t_{k,n} - t_n^0) \frac{N_v}{r} - \psi_v(t_{k,n}). \quad (15)$$

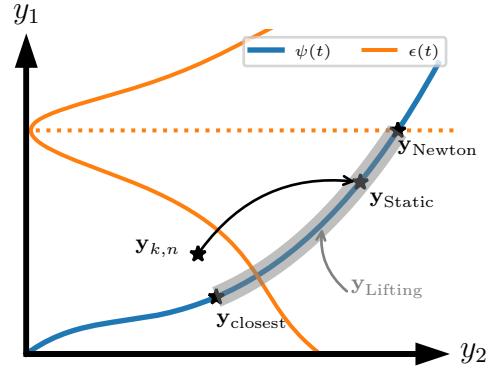
This residual measures the time deviation between the requested projection time  $t_{k,n}$ , and the time corresponding to the resulting image row,  $\psi_v$ . We choose to express this deviation in rows (pixels), instead of time (seconds), because this makes it easier to compare it to the reprojection error.

An ideal rolling shutter projection model should always fulfill the *rolling shutter constraint*

$$\epsilon(t_{k,n}) = 0, \quad (16)$$

but we will see that relaxing this constraint can result in other benefits, while still producing reasonable results.

In Figure 5 we graphically compare the three different methods, by plotting their possible image projections,  $\psi(t_{k,n})$ , together with the time deviation  $\epsilon(t_{k,n})$ .



**Figure 5.** Geometric illustration of the different approaches to the projection time problem. This is an image plane plot, where  $y_1$  is the rolling shutter axis,  $y_{k,n}$  is the landmark observation in the current frame,  $\psi(t)$  is the reprojeciton (transfer) curve for the first observation, as a function of spline evaluation time  $t$ , and  $\epsilon(t)$  is the absolute value of the projection time deviation, plotted along the  $y_1$  axis, as a function of  $t$ . In the illustration we can see that  $y_{Static}$  is obtained by setting the spline time to the observation time of the landmark  $y_{k,n}$ ,  $y_{Newton}$  is the point on the reprojeciton curve  $\psi(t)$  that perfectly satisfies the projection time constraint  $\epsilon(t)$ , and  $y_{Lifting}$  is a point on  $\psi(t)$  somewhere between  $y_{closest}$ , the point closest to the observation, and  $y_{Newton}$  (depending on residual weighting).

### 3.2 Static projection

One simple approach to deal with the chicken and egg problem described in section 3, is to ignore it completely. If we denote the observed image row by  $v_{k,n}$ , we set the projection time to

$$t_{k,n} = t_n^0 + r \frac{v_{k,n}}{N_v} \quad (17)$$

and directly compute (14).

The advantage of this method is that it is fast to compute, and simple to implement. The downside is that the projected point in general will not fulfill the rolling shutter constraint in (16). This is shown in Figure 5, where the  $y_{Static}$  point can end up anywhere on the  $\psi(t)$  line, regardless of the value of  $\epsilon(t)$ .

### 3.3 Newton projection

To make sure that the rolling shutter projection time constraint in (16) holds, Patron-Perez et al. (2015) uses Newton's method to iteratively find the projection time.

To use Newton's method to solve  $\epsilon(t) = 0$  we must compute  $\frac{d\epsilon(t)}{dt}$ , which in turn requires computation of  $\frac{d\psi(t)}{dt}$ . The transfer function  $\psi(t)$  involves applying the camera projection model,  $\pi(t)$ , and its inverse,  $\pi^{-1}(t)$ , which means that the implementation can be quite tricky, as derivatives of these functions are also required. Each iteration is thus more expensive than the **Static** method, but we must also compute multiple iterations, making this a quite slow strategy. The advantage is of course that the rolling shutter time constraint (16) is now fulfilled, as we can see in Figure 5.

### 3.4 Lifting

The two previous methods are extremes when it comes to trading accuracy for computational efficiency. Kim et al. (2016) therefore introduced a third method that aims to be more accurate than **Static**, while being faster than **Newton**.

This works by adding the time deviation  $\epsilon(t_{k,n})$  (see (15)) as a new residual to the optimization problem. The unknown projection time  $t_{k,n}$  is now an additional parameter to optimize over.

The added residual makes (16) into a soft constraint, which means that at best it will match the **Newton** method, and at worst give the point closest to the measured observation. See Figure 5 for a graphical illustration.

The described method, which we denote **Lifting**, has the same computational complexity as the **Static** method. However, since we are adding an extra residual and parameter per image observation, the optimization problem grows larger.

## 4 Spline interpolation spaces

A time-continuous pose  $\mathbf{T}(t)$  consists of a rotational component  $\mathbf{R}(t)$ , and a translational component  $\mathbf{p}(t)$ ,

$$\mathbf{T}(t) = \begin{bmatrix} \mathbf{R}(t) & \mathbf{p}(t) \\ \mathbf{0}^T & 1 \end{bmatrix}. \quad (18)$$

Nearly all continuous camera pose representations are based on B-splines, that define the continuous pose by blending discrete poses  $\{\mathbf{T}_k\}_1^K$ . In this section we introduce and compare the two trajectory representations that are used in this work: one interpolating over control points  $\mathbf{T}_k \in \mathbb{SE}(3)$ , and one that uses separate splines for translation, and rotation, with control points  $\mathbf{p}_k \in \mathbb{R}^3$ , and  $\mathbf{R}_k \in \mathbb{SO}(3)$ , respectively. We also analyze the theoretical difference between the two when interpolating a camera pose.

### 4.1 A split spline in $\mathbb{R}^3$ and $\mathbb{SO}(3)$

A regular B-spline curve in vector space  $\mathbb{R}^n$  can be written:

$$\mathbf{p}(t) = \sum_{k=1}^K \mathbf{p}_k B(t - k\Delta t) = \sum_{k=1}^K \mathbf{p}_k B_k(t), \quad (19)$$

where  $\mathbf{p}_k \in \mathbb{R}^n$  are the spline control points, and  $B_k(\cdot)$  are the shifted B-spline basis functions (cf. (6)), that distribute the influence of each control point in a specific time window.

Any spline of form (19) may instead be written in cumulative form:

$$\mathbf{p}(t) = \mathbf{p}_1 \tilde{B}_1(t) + \sum_{k=2}^K (\mathbf{p}_k - \mathbf{p}_{k-1}) \tilde{B}_k(t), \quad (20)$$

where  $\tilde{B}(t)$  are the corresponding *cumulative* basis functions. Kim et al. (1995) show that this construction is also feasible on  $\mathbb{SO}(3)$ , and propose to use unit quaternions  $\mathbf{q}_k$  as orientation control points to interpolate

$$\mathbf{q}(t) = \mathbf{q}_1^{\tilde{B}_1(t)} \prod_{k=2}^K \exp(\log(\mathbf{q}_{k-1}^* \mathbf{q}_k) \tilde{B}_k(t)). \quad (21)$$

Here  $\mathbf{q}^*$  denotes the conjugation of the quaternion  $\mathbf{q}$ , and  $\exp()$  and  $\log()$  are mappings to  $\text{Spin}(3)$ , and its tangent

space, respectively. The rationale behind (21) is the classical SLeRP interpolation (Shoemake 1985):

$$\mathbf{q}(\lambda) = \mathbf{q}_1 \exp(\lambda \log(\mathbf{q}_1^* \mathbf{q}_2)) \quad \lambda \in [0, 1]. \quad (22)$$

The expression (22) moves smoothly between  $\mathbf{q}_1$  and  $\mathbf{q}_2$  as  $\lambda$  is moved from 0 to 1. By comparing (21) with (22) we see that the Kim et al. construction is essentially a blending of SLeRP interpolations, within each B-spline support window.

In summary, Kim et al. (1995) advocate pose interpolation with (20) for position and (21) for orientation. We will denote this as *split interpolation*, or *split representation*.

**4.1.1 IMU predictions for the split interpolation.** The IMU predictions for the split representation is most suitably derived using quaternion algebra, with vectors  $\mathbf{v} \in \mathbb{R}^3$  embedded in pure quaternions  $\mathbf{q}_v = (0 \ \mathbf{v})^T$ .  $\mathbf{g}$  is the gravity vector, in the global coordinate frame. We only show how to get the ideal gyroscope and IMU measurements from the trajectory, and disregard other aspects of the IMU model, such as bias, or axis misalignment.

### Gyroscope prediction

$$\begin{pmatrix} 0 \\ \nabla_\omega \mathbf{T}(t) \end{pmatrix} = \mathbf{q}_\omega^{\text{body}}(t) = \mathbf{q}^*(t) \mathbf{q}_\omega^{\text{global}}(t) \mathbf{q}(t) \quad \text{where} \quad (23)$$

$$\mathbf{q}_\omega^{\text{global}}(t) = 2\dot{\mathbf{q}}(t)\mathbf{q}^*(t) \quad (24)$$

### Accelerometer prediction

$$\begin{pmatrix} 0 \\ \nabla_a^2 \mathbf{T}(t) \end{pmatrix} = \mathbf{q}^*(t) \begin{pmatrix} 0 \\ \ddot{\mathbf{p}}(t) - \mathbf{g} \end{pmatrix} \mathbf{q}(t) \quad (25)$$

### 4.2 A spline in $\mathbb{SE}(3)$

In Patron-Perez et al. (2015) the quaternion spline (21) is generalized to a spline construction with control points  $\mathbf{T}_k \in \mathbb{SE}(3)$ :

$$\mathbf{T}(t) = \exp(\log(\mathbf{T}_1) \tilde{B}_1) \prod_{k=2}^K \exp(\log(\mathbf{T}_{k-1}^{-1} \mathbf{T}_k) \tilde{B}_k(t)). \quad (26)$$

Just like in the quaternion case, this is a blend of linear interpolations on the group, within each B-spline window.

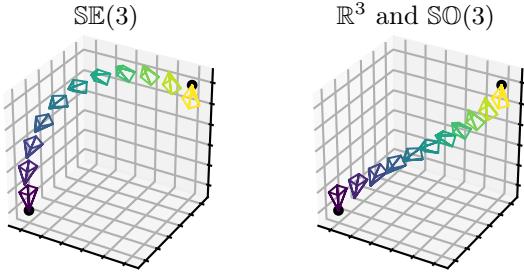
In Patron-Perez et al. (2015) the poses to interpolate are defined as transformations from the body frame to the global frame, i.e.,

$$\mathbf{T}(\mathbf{R}, \mathbf{p}) = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}^T & 1 \end{bmatrix}, \quad (27)$$

where  $\mathbf{p}$  is the spline position in the global frame, and  $\mathbf{R}$  is the rotation from the body frame to the global frame. Note that interpolation of  $\mathbf{p}$  and  $\mathbf{R}$  separately, using (20) and (21) is not equivalent to (26). The difference between the two is revealed by expanding the  $\mathbb{SE}(3)$  tangent, or *twist* (Murray et al. 1994), that is used to move between two poses in (26):

$$\log(\mathbf{T}_1^{-1} \mathbf{T}_2) = \log \begin{bmatrix} \mathbf{R}_1^T \mathbf{R}_2 & \mathbf{R}_1^T (\mathbf{p}_2 - \mathbf{p}_1) \\ \mathbf{0}^T & 1 \end{bmatrix}. \quad (28)$$

A twist  $\xi = (\mathbf{v}, \boldsymbol{\omega}) \in \mathfrak{se}(3)$ , consists of a translation  $\mathbf{v}$  (with direction and scale), and an axis-angle vector  $\boldsymbol{\omega}$ . By



**Figure 6.** Trajectories from interpolation of two poses on  $\mathbb{SE}(3)$  (left) and separate interpolation in  $\mathbb{R}^3$  and  $\mathbb{SO}(3)$  (right). Here, start and end poses differ by  $150^\circ$  in orientation, which exposes the screw motion caused by the  $\mathbb{SE}(3)$ -based interpolation.

exponentiating a twist times a scalar amount  $\theta$  we obtain an element in  $\mathbb{SE}(3)$ , with the following analytic expression:

$$\exp(\xi\theta) = \exp\left(\begin{bmatrix} [\omega]_\times & \mathbf{v} \\ \mathbf{0}^T & 0 \end{bmatrix}\theta\right) = \begin{bmatrix} \exp([\omega]_\times\theta) & (\mathbf{I} - \exp([\omega]_\times\theta))[\omega]_\times\mathbf{v} + \omega\omega^T\mathbf{v}\theta \\ \mathbf{0}^T & 1 \end{bmatrix}, \quad (30)$$

where  $[\cdot]_\times$  is the cross product operator, i.e.,  $[\mathbf{a}]_\times\mathbf{b} = \mathbf{a} \times \mathbf{b}$ , see (Murray et al. 1994, eq. 2.36). In analogy with this, the twist in (28) is weighted by a basis function value  $\tilde{B}_k(t)$  and exponentiated in (26). We can thus identify  $\theta$  with  $\tilde{B}_k(t)$ .

**4.2.1 IMU predictions for  $\mathbb{SE}(3)$ .** To compute the IMU predictions for  $\mathbb{SE}(3)$ , we use the same formulation as in Patron-Perez et al. (2015). Here  $\dot{\mathbf{R}}(t)$ ,  $\dot{\mathbf{p}}(t)$ , and  $\ddot{\mathbf{p}}(t)$ , are the corresponding submatrices of  $\dot{\mathbf{T}}(t)$ , and  $\ddot{\mathbf{T}}(t)$ .  $\mathbf{g}$  is the gravity vector, in the global coordinate frame. Again, we only show how to get the ideal gyroscope and IMU measurements from the trajectory, and disregard other aspects of the IMU model.

### Gyroscope prediction

$$\nabla_\omega \mathbf{T}(t) = \boldsymbol{\omega} \text{ where} \quad (31)$$

$$[\boldsymbol{\omega}]_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} = \mathbf{R}^T(t)\dot{\mathbf{R}}(t) \quad (32)$$

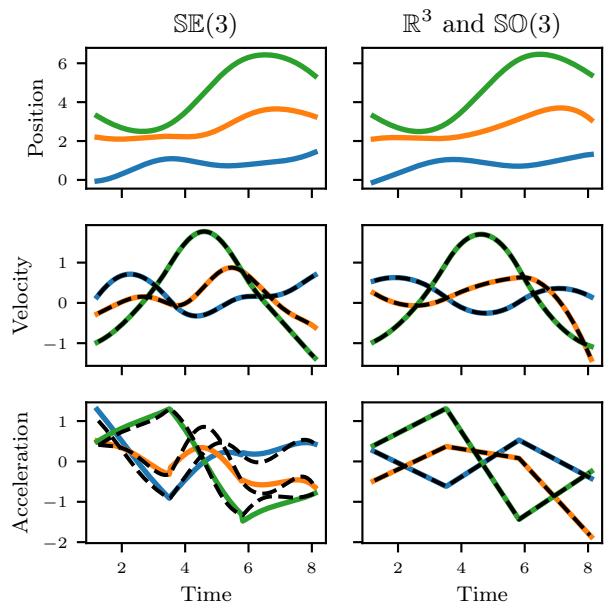
### Accelerometer prediction

$$\nabla_a^2 \mathbf{T}(t) = \mathbf{R}^T(t)(\ddot{\mathbf{p}}(t) - \mathbf{g}) \quad (33)$$

### 4.3 Why $\mathbb{SE}(3)$ splines are problematic

Here we describe a number of problems with choosing  $\mathbb{SE}(3)$  as the interpolation space.

**4.3.1 Translation is linked with orientation.** By identifying the exponentiation of (28) with (30), when  $\theta = 1$ , we can further identify the rotation component as  $\exp([\boldsymbol{\omega}]_\times) = \mathbf{R}_1^T \mathbf{R}_2$  (and thus  $\boldsymbol{\omega}$  is parallel to the axis of rotation, which implies  $\boldsymbol{\omega} = \mathbf{R}_1^T \mathbf{R}_2 \boldsymbol{\omega}$ ). For intermediate values of  $\theta$ , the translation in (30) consists of a component parallel to the rotation axis (i.e.,  $\boldsymbol{\omega}\boldsymbol{\omega}^T\mathbf{v}$ ) and one orthogonal to it (i.e.,



**Figure 7.** The problem with acceleration in  $\mathbb{SE}(3)$ . Solid and colored lines are the position, velocity, and acceleration as computed from the spline interpolation. Black dashed lines are the same quantities, but instead computed by numerical differentiation.

$[\boldsymbol{\omega}]_\times \mathbf{v}$ ) that depends on the amount of rotation. Unless the translation is parallel to the rotation axis, there will thus be an interaction between the rotation and the translation. The effect of this coupling of translation and orientation is that the camera position moves along a trajectory that spirals about the rotation axis  $\boldsymbol{\omega}$ , as exemplified in Figure 6. Such a motion is called a *screw motion* (Murray et al. 1994).

The implicit mechanical model in  $\mathbb{SE}(3)$ -based interpolation is that the pose is manipulated by an *internal force and torque*, i.e., a force applied to the same fixed reference point, and with a torque about a fixed axis in the intrinsic pose frame (such an action is called a *wrench* (Murray et al. 1994)). For separate interpolation of position and orientation (see section 4.1), pose is instead manipulated by a *generic force and torque* acting on the pose frame in different ways at different times.

The above interpretation predicts that the  $\mathbb{SE}(3)$  model would be a good fit for e.g., cameras mounted at the end of a robot arm, and in the idealized case also car mounted cameras e.g., dashcams. The split interpolation model makes fewer assumptions about how the motion changes, and is thus likely to be of more general use.

**4.3.2 Derivative vs. body acceleration.** To compute the accelerometer predictions, (33) and (25), we must first compute the linear acceleration of the body, denoted  $\ddot{\mathbf{p}}(t)$ . For split interpolation this is simply the second order derivative of the  $\mathbb{R}^3$ -spline, which does not impose any problem. Using the  $\mathbb{SE}(3)$  representation,  $\ddot{\mathbf{p}}(t)$  is defined as a submatrix of  $\ddot{\mathbf{T}}(t)$ . In section 4.3.1 we saw that during interpolation,  $\mathbb{SE}(3)$  introduces a dependency between the translation part, and the orientation part. It turns out that a similar orientation-dependency problem exists also when

computing the acceleration, see e.g. Zefran and Kumar (1996).

We illustrate this problem in Figure 7. Here we have constructed two pose trajectories: one in  $\mathbb{SE}(3)$  and one split in  $\mathbb{R}^3$  and  $\mathbb{SO}(3)$ . These trajectories have equal knot placements, and are designed to be as similar as possible. For a trajectory to be well behaved, we expect that its velocity is the first order derivative of the position ( $\mathbf{v}(t) = \frac{d\mathbf{p}(t)}{dt}$ ), and that acceleration is the first order derivative of velocity ( $\mathbf{a}(t) = \frac{d\mathbf{v}(t)}{dt}$ ). To test whether this holds true for the two trajectories, we first analytically compute the position, velocity, and acceleration, using their respective spline formulations (Patron-Perez et al. 2015, eqns. 4-6). We then compute velocity and acceleration again, but this time using numerical differentiation of position and velocity, respectively. The idea is to now check whether the numerical and analytical quantities are equal.

Figure 7 clearly shows that both trajectory representations behave as expected with respect to velocity, since the analytical and numerical results are identical. For acceleration, we can see that this holds true only for the split interpolation, while  $\mathbb{SE}(3)$  shows severe differences. Only if we were to set the orientation constant ( $\mathbf{R}(t) = \mathbf{R}$ ) does the analytical and numerical results agree, which verifies that the problem is indeed caused by interaction with the orientation.

This means that the acceleration produced by the  $\mathbb{SE}(3)$  trajectory derivative is not the true, kinematic, body acceleration. Accelerometer predictions computed from it, will therefore also be inaccurate.

**4.3.3 Efficiency.** In general, evaluation of an  $\mathbb{SE}(3)$  spline is slightly more expensive, as the translation part of a spline is evaluated using the upper right element in (30) instead of the simpler vector difference in (20). The  $\mathbb{SO}(3)$  part is, however, the same for both methods.

Another efficiency issue has to do with evaluation of derivatives. Here, the split  $\mathbb{R}^3$  and  $\mathbb{SO}(3)$  representation allows for a potential speedup by choosing to compute only the derivatives that are required for each term in the visual-inertial cost function (11):

- To compute the gyroscope residuals (see (24) and (32)), only the first order orientation derivative is needed. However, when using  $\mathbb{SE}(3)$  we must compute the full  $\dot{\mathbf{T}}(t)$  matrix, which implicitly also calculates the superfluous linear part.
- Computing the acceleration residuals (see (25) and (33)) requires the linear acceleration, and orientation. In the case of split interpolation on  $\mathbb{R}^3$  and  $\mathbb{SO}(3)$ , the linear acceleration in  $\mathbb{R}^3$  is very efficient to compute, while we only need to evaluate the orientation in  $\mathbb{SO}(3)$ . In  $\mathbb{SE}(3)$ , we must of course compute the full  $\ddot{\mathbf{T}}(t)$  matrix, which requires more computations.

## 5 Experiments

In section 4 we described two different choices of trajectory representation, and their properties and theoretical problems. We will now investigate what impact the identified problems have on practical applications. In section 3, we described three different choices of rolling shutter projection methods. We now want to see how these methods differ with respect

to accuracy and runtime efficiency. To investigate this, we perform a number of experiments on both synthetically generated, and recorded real data.

### 5.1 Software

To estimate the trajectory and 3D structure we used the open source *Kontiki* framework (Ovrén 2018), which is developed by us<sup>1</sup>. Kontiki is a general purpose continuous-time trajectory estimation framework, built to be easy to extend. Users choose a trajectory, add measurements (IMU, camera, etc.), and then ask the framework to find the most probable trajectory matching the measurements. The least-squares solver uses the well known *Ceres Solver* (Agarwal et al. 2012–2018), and for  $\mathbb{SE}(3)$  calculations we use the *Sophus* library (Strasdat and Lovegrove 2011–2017). Kontiki is written in C++, but is mainly intended to be used with its Python frontend.

### 5.2 Reconstruction method

All experiments follow the same reconstruction pipeline, which we describe here.

First we compute a suitable knot spacing for the splines, using the method by Ovrén and Forssén (2018), summarized in section 2.6.2. Since that method assumes a split spline defined on  $\mathbb{R}^3$  and  $\mathbb{SO}(3)$ , we get one knot spacing for each interpolation space:  $\Delta t_{\mathbb{R}^3}$  and  $\Delta t_{\mathbb{SO}(3)}$ . To make the comparison with  $\mathbb{SE}(3)$  fair, we set  $\Delta t = \min(\Delta t_{\mathbb{R}^3}, \Delta t_{\mathbb{SO}(3)})$ , and use this value for *all* splines. From the selected knot spacing,  $\Delta t$ , we then computed the corresponding IMU norm weights,  $\mathbf{W}_a$  and  $\mathbf{W}_g$ , as summarized in section 2.6.1.

Like Ovrén and Forssén (2018), we use keyframing to reduce the number of measurements, to reduce the processing time. In this case, we extract keyframes uniformly, spaced 10 frames apart. We then use the adaptive non-maxima suppression method by Gauglitz et al. (2011) to select the set of landmarks and observations such that each keyframe has at most 100 observations.

Trajectories are initialized such that  $\mathbf{p}(t) = \mathbf{0}$ , and  $\mathbf{R}(t) = \mathbf{I}$ , for all  $t$ . Landmarks are set to points at infinity, using  $\rho_k = 0$ .

The robust error norm  $\phi(\cdot)$  is the Huber norm, with parameter  $c = 1$ .

### 5.3 Datasets

To show that the optimization behaves differently depending on the choice of interpolation space we define the following types of motion that we want to investigate:

1. **Free.** Camera with free orientation. The camera orientation changes independently of the motion path. This simulates the case of a handheld camera, or a camera mounted on a gimbal on e.g., a UAV.
2. **Forward.** Camera locked in the forward direction of the path. This is similar to e.g., a dash-cam mounted in a car.
3. **Sideways.** As above but the camera is mounted looking 90° left or right.



**(a)** The GoPro camera with attached IMU logger      **(b)** The radio controlled car used for the **RC-Car** dataset

**Figure 8.** Hardware used for experiments

Checking both the **Forward locked** and **Sideways locked** cases are of interest since they are known to differ in difficulty, where the former is harder (Vedaldi et al. 2007).

**5.3.1 Synthetic data.** Our synthetic data was created using the *IMUSim* software package (Young et al. 2011). Since IMUSim only models IMU measurements, we implemented an extension package<sup>2</sup> that models rolling shutter cameras.

For each of the motion types we generated 200 random trajectories, with matching 3D-structure, which were then processed by the simulator. For the **Forward** and **Sideways** cases the ground truth trajectories were generated using a simple motion model that tried to emulate a driving car.

The landmarks were projected into the simulated camera by finding a solution for  $\epsilon(t_{k,n}) = 0$ , using the bounded root search method by Brent (1973).

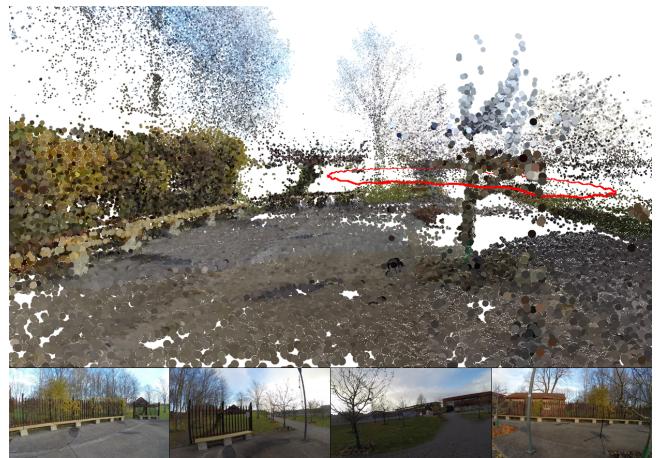
**5.3.2 Real data.** For the real data experiments we used two datasets<sup>3</sup> called **Handheld** and **RC-Car**.

For both datasets, we used a *GoPro Hero 3+ Black* camera, to which we attached a custom designed IMU logger, see Figure 8a. The camera was recording using 1080p wide mode at 29.97 Hz, while the IMU measurements were collected at 1000 Hz. In the experiments, the raw IMU samples were resampled to 300 Hz to reduce processing time.

The **Handheld** dataset was recorded while holding the camera and walking in a loop outdoors. Since the camera was free to rotate, it represents the **Free** motion type. Example frames from the **Handheld** dataset can be found in Figure 9.

In the **RC-Car** dataset, the camera was attached to a radio controlled car, see Figure 8b. The camera was mounted pointing forwards, and thus represents the **Forward** motion type. The RC-car was then driven in a loop over (relatively) rough terrain, resulting in both high-frequency motion and motion blur. Example frames from the **RC-Car** dataset can be found in Figure 1.

Image measurements were collected by tracking FAST features (Rosten et al. 2010) over subsequent frames, using the OpenCV KLT-tracker (Bouguet 2000). For added robustness, we performed backtracking and discarded tracks which did not return to within 0.5 pixels of its starting point. Using tracking instead of feature matching means that landmarks that are detected more than once will be tracked multiple times by the system.



**Figure 9.** Rendered model estimated on the **Handheld** dataset, using split interpolation. Top: model rendered using Meshlab. Bottom: Sample frames from dataset.

The camera-IMU extrinsics, gyroscope bias, and time offset, were given an initial estimate using the *Crisp* (Ovrén and Forssén 2015) toolbox. Since Crisp does not support accelerometer measurements, we then refined the initial estimate using Kontiki, described in section 5.1, by optimizing over a short part of the full sequence with the accelerometer bias as a parameter to optimize.

#### 5.4 Trajectory representation convergence rates

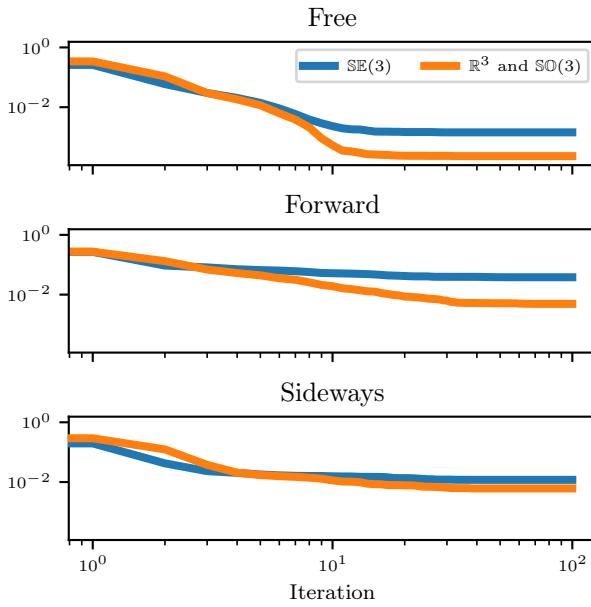
We want to investigate whether the choice of trajectory representation has any impact on the reconstruction process. By performing many reconstructions using both trajectory representations, we can gather statistics on how the optimization cost changes over time. Ideally we would like to compare reconstruction quality, but since the real dataset does not have any ground truth, this is not possible. The use of convergence rate as a metric is thus justified by the fact that it allows us to compare the results from the synthetic and the real datasets. Since a failed reconstruction should also cause a higher cost, the reconstruction quality is implicitly measured by the convergence metric.

In order to gather statistics also for the real datasets (of which we have only two, longer, sequences), we split them into a set of five seconds long, overlapping, slices, and perform the reconstructions on these instead.

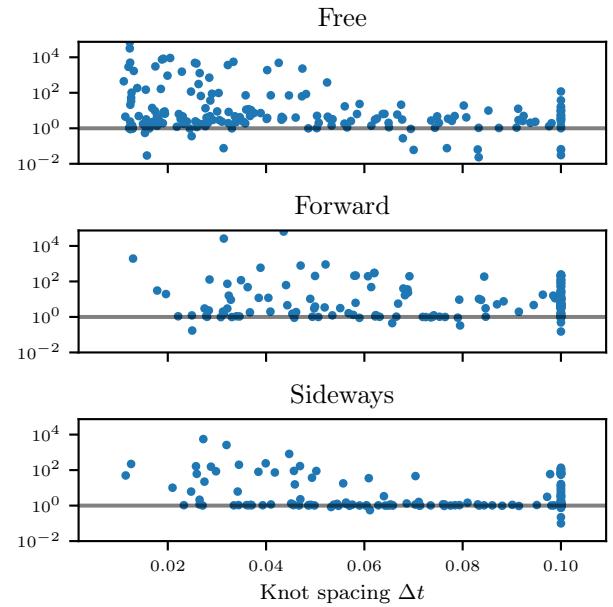
In the synthetic datasets, the camera observations and IMU measurements were perturbed by additive gaussian noise with  $\sigma_{\text{image}} = 0.5$  and  $\sigma_{\text{IMU}} = 0.01$ , respectively.

We always used exactly the same measurements for the  $\mathbb{SE}(3)$  and split reconstructions.

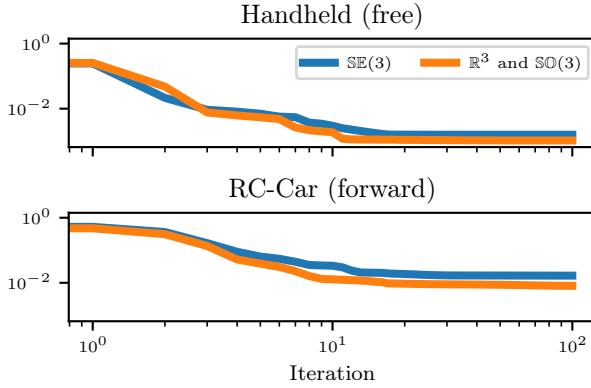
Figures 10 and 11 show the median *relative cost* per iteration for the synthetic and real datasets, respectively. The relative cost is simply the ratio between the current iteration cost, and the initial cost at iteration 0. To give an idea on the distribution, the shaded area shows the 40/60 percentiles of the data. We can see that the split trajectory performs much better than  $\mathbb{SE}(3)$ , giving a larger reduction in cost, which indicates a better solution. This is true both for the synthetic and real data case, and for all motion types.



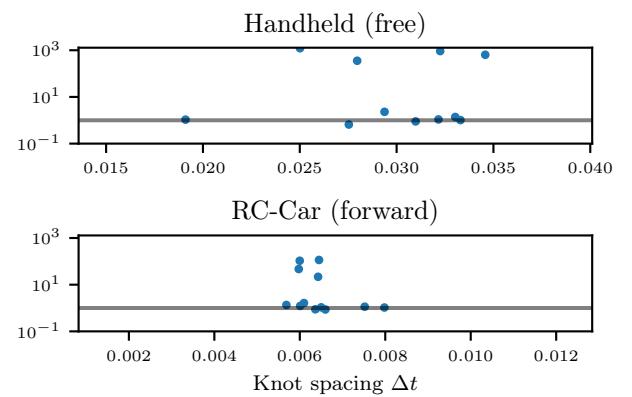
**Figure 10.** Convergence rate results on the synthetic dataset. The Y-axis shows the ratio between the current iteration cost and the initial cost at iteration 0. Solid line is the median, and the shaded area shows the distribution using the 40/60-percentiles.



**Figure 12.** Distribution of relative performance between split interpolation on  $\mathbb{R}^3$  and  $\text{SO}(3)$  and  $\text{SE}(3)$  on synthetic data. The Y-axis shows the ratio between their respective relative costs at the final iteration. Samples above the line are where split representation performed better.



**Figure 11.** Convergence rate results on the real dataset. The Y-axis shows the ratio between the current iteration cost and the initial cost at iteration 0. Solid line is the median, and the shaded area shows the distribution using the 40/60-percentiles.



**Figure 13.** Distribution of relative performance between split interpolation on  $\mathbb{R}^3$  and  $\text{SO}(3)$  and  $\text{SE}(3)$  on real data. The Y-axis shows the ratio between their respective relative costs at the final iteration. Samples above the line are where split representation performed better.

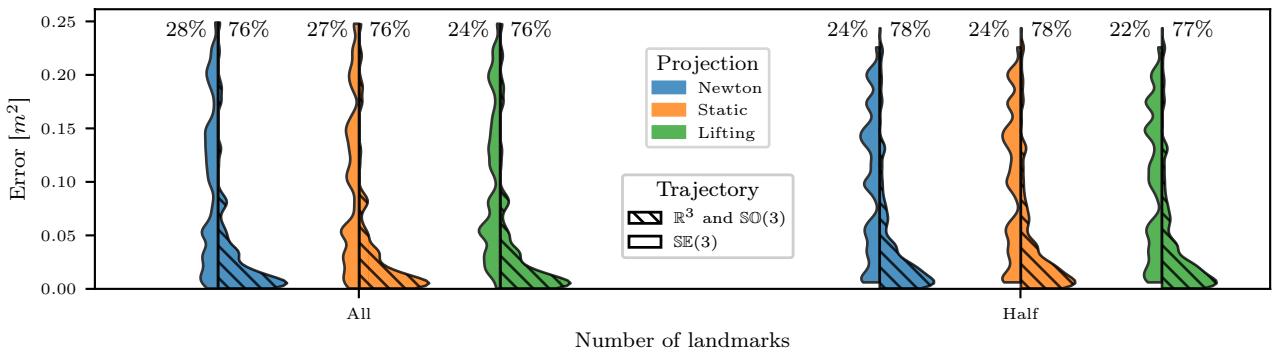
In section 4.3.1 we hypothesized that  $\text{SE}(3)$  could be a better choice for the fixed orientation cases. It is clear from Figure 10 that the difference between split interpolation and  $\text{SE}(3)$  is largest on the **Free** dataset, which corroborates this. However,  $\text{SE}(3)$  is clearly inferior on *all* datasets, both real and synthetic, which means that the negative aspects of  $\text{SE}(3)$ , as described in section 4.3, outweigh the possible benefit this might have had.

To get further clues on what might affect performance, we plot the relative cost ratio for each reconstruction as a function of the chosen knot spacing. As we can see in figures 12 and 13 it is clear that  $\text{SE}(3)$  tends to have worse performance for small knot spacings (denser splines).

## 5.5 Projection method

In section 3 we described three different methods to do rolling shutter landmark projection. Since they differ both in implementation complexity, and runtime efficiency, we want to make sure that slower and more complex methods actually result in better accuracy.

In this experiment we performed reconstructions on the 200 sequences in the **Free** dataset, for all combinations of trajectory representations and projection methods. To also investigate whether any of the methods are sensitive to the available amount of data, we also performed the reconstructions with only half the available landmarks.



**Figure 14.** Distribution of errors for different combinations of trajectory representation and landmark projection method. The violin plots show the distribution for all errors in the inlier set, for which the error  $< 0.25$ . The percentage above each violin is the inlier ratio.

To evaluate the result we compared the estimated trajectory to the ground truth trajectory using the soap-bubble area between the two position trajectories, as previously suggested in [Hedborg et al. \(2012\)](#). The optimized trajectory was aligned to the ground truth by using Horn’s method for the translation ([Horn 1987](#)) and the orthogonal Procrustes method for the orientation ([Golub and van Loan 1983](#)). Since the optimization gives a trajectory with a metric scale, we do not need to estimate any scaling factor, as was done in [Hedborg et al. \(2012\)](#). For two position trajectories  $\mathbf{f}(t)$  and  $\mathbf{g}(t)$ , we compute the area error numerically by trapezoid summation:

$$a(\mathbf{f}, \mathbf{g}) = \sum_{k=1}^{K-1} a_{\text{trap}}(\mathbf{f}(t_k), \mathbf{f}(t_{k+1}), \mathbf{g}(t_k), \mathbf{g}(t_{k+1})), \text{ where} \quad (34)$$

$$a_{\text{trap}}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = \frac{\|\mathbf{a} - \mathbf{c}\|}{2} (\|\mathbf{a} - \mathbf{b}\| + \|\mathbf{c} - \mathbf{d}\|). \quad (35)$$

This approximation of the area is only valid when sampled densely, which we do.

In Figure 14 we plot the error distributions for all tested combinations. Since some reconstructions fail, we choose to plot only an inlier set, which we define as the samples with an error below  $0.25\text{m}^2$ . The results in Figure 14 support the conclusion from the convergence experiment in section 5.4:  $\text{SE}(3)$  fails more often than split interpolation, as shown by the inlier percentage. However, even for the inlier sets, it is clear that split interpolation provides better reconstructions since most of the distribution mass is concentrated at lower errors.

Looking only at the results for split interpolation we can see that all three projection methods perform more or less identically. Also, they all benefit from more available data, which is expected.

## 5.6 Efficiency

The choice of interpolation space and reprojection method will affect the runtime of the optimization. In Table 1 we show the mean iteration time of our implementations on the **Free** dataset, normalized with respect to  $\text{SE}(3)$  with **Newton**. Note that these timings include also the time to compute the IMU residuals.

**Table 1.** Mean iteration time for different choices of interpolation space and projection method. The times are given relative to  $\text{SE}(3)$  with **Newton**. Lower values are faster.

	Newton	Static	Lifting
$\text{SE}(3)$	1.00	0.52	0.54
$\mathbb{R}^3$ and $\text{SO}(3)$	0.54	0.36	0.37

From section 4.3.3 we hypothesized that  $\text{SE}(3)$  would be the slowest, both because it is more computationally involved, and because it must compute superfluous derivatives for the IMU measurements. In our implementations, split interpolation on  $\mathbb{R}^3$  and  $\text{SO}(3)$  is roughly twice as fast as  $\text{SE}(3)$  per iteration, which supports this.

The **Static** and **Lifting** reprojection methods share the same cost function, but the latter adds parameters to the optimization which should yield a higher per-iteration cost. The cost of the **Newton** method is linear in the number of iterations taken, which is usually around 2.

Although performance is always contingent upon the specific implementation, these practical results are consistent with the principled discussion above. Also, the  $\text{SE}(3)$  (i.e., Sophus by [Strasdat and Lovegrove \(2011–2017\)](#)) and split implementations both use the same *Eigen* ([Guennebaud et al. \(2010\)](#)) linear algebra library for spline interpolation and projection computations, ensuring a fair comparison.

## 5.7 Example reconstructions

To showcase the real dataset, and to also verify that reconstruction is possible, we performed 3D reconstruction on the original, full, sequences. We used the pipeline from [Ovrén and Forssén \(2018\)](#), which uses a split trajectory and the **Static** projection method.

Since the resulting sparse reconstructions are hard to visualize, we densified them by triangulating new landmarks using the final trajectory. During this densification step, the trajectory was locked, and not updated. The trajectories and densified 3D scenes are shown in figures 1 and 9.

## 6 Conclusions and future work

We have looked at two different spline-based trajectory representations, and compared them theoretically, and experimentally. From the presented theory we hypothesized that  $\text{SE}(3)$  interpolation would perform worse than split interpolation because it makes translation dependent on the orientation. The experiments support this, since the  $\text{SE}(3)$  spline converges slower, and to a worse result than interpolation on  $\mathbb{R}^3$  and  $\text{SO}(3)$ , while also having a much higher failure rate. It is also clear that  $\text{SE}(3)$  is less efficient, being roughly half as fast as split interpolation. A split  $\mathbb{R}^3$  and  $\text{SO}(3)$  spline also has the added flexibility of allowing splines of different densities. Because of these findings, we recommend that researchers use a split  $\mathbb{R}^3$  and  $\text{SO}(3)$  spline over an  $\text{SE}(3)$  spline for this type of application.

The three landmark projection methods all performed well, and produced nearly identical results. There was however a large difference in efficiency, with **Newton** up to twice as slow as **Lifting** and **Static**. In the context of continuous-time structure from motion, we therefore recommend researchers to use the **Static** projection method, since it is both the fastest, and the most easy to implement. In other applications, e.g., when the rolling shutter readout time is also calibrated for (Oth et al. 2013), the difference between the methods may be larger. Here, hybrid optimization schemes could be of interest, where a fast method is used initially, and a more accurate one is used in the final few iterations.

In the experiments all reconstructions were started with a trajectory with constant position  $\mathbf{p}(t) = \mathbf{0}$  and orientation  $\mathbf{R}(t) = \mathbf{I}$ , and landmarks at infinity with  $\rho_k = 0$ . In contrast, discrete-time structure from motion requires a suitable initialization to get any meaningful result. We believe that this works because the addition of inertial measurements gives constraints on the shape of the trajectory which can force even a bad starting state into something useful. From the experiments it is clear that while this initialization-free start works quite well in general (at least for a spline defined on  $\mathbb{R}^3$  and  $\text{SO}(3)$ ), there are failure cases. In the future we would like to investigate more robust ways to perform initialization for visual-inertial fusion. On the synthetic **Forward** and **Sideways** datasets, we have observed a correlation between the velocity of the simulated vehicle, and the final relative cost value. We hypothesize that the lack of a zero-velocity point makes the estimation harder, since the integration from accelerometer measurements to velocity assumes an initial speed of 0. If available, adding velocity measurements to the optimization could be a way to remedy this.

## Acknowledgements

The authors would like to thank Andreas Robinson for designing the IMU logger, and helping out with the radio controlled car.

## Funding

This work was funded by the Swedish Research Council through projects LCMM (2014-5928) and EMC2 (2014-6227).

## Notes

1. Kontiki will be released to the public in June 2018.
2. The rolling shutter extension to IMUSim can be found at <https://github.com/hovren/rsimusim>.
3. The full dataset is available from <http://www.cvl.isy.liu.se/research/datasets/gopro-imu-dataset/>

## References

- Agarwal S, Mierle K and Others (2012–2018) Ceres solver. <http://ceres-solver.org>.
- Anderson S and Barfoot TD (2015) Full STEAM ahead: Exactly sparse gaussian process regression for batch continuous-time trajectory estimation on  $\text{SE}(3)$ . In: *IEEE International Conference on Intelligent Robots and Systems (IROS15)*.
- Bailey T and Durrant-Whyte H (2006) Simultaneous localization and mapping (SLAM): part II. *IEEE Robotics & Automation Magazine* 13(3): 108–117. DOI:10.1109/MRA.2006.1678144.
- Bouguet JY (2000) Pyramidal implementation of the Lucas Kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*.
- Brent RP (1973) *Algorithms for Minimization without Derivatives*, chapter 4: An Algorithm with Guaranteed Convergence for Finding a Zero of a Function. Englewood Cliffs, NJ: Prentice-Hall.
- Crouch P, Kun G and Leite FS (1999) The de Casteljau algorithm on lie groups and spheres. *Journal of Dynamical and Control Systems* 5(3): 397–429.
- Engel J, Koltun V and Cremers D (2016) Direct sparse odometry. In: *ArXiv:1607.02565*.
- Forster C, Carbone L, Dellaert F and Scaramuzza D (2015) IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In: *Robotics: Science and Systems (RSS'15)*. Rome, Italy.
- Furgale P, Barfoot TD and Sibley G (2012) Continuous-time batch estimation using temporal basis functions. In: *IEEE International Conference on Robotics and Automation (ICRA12)*.
- Furgale P, Tong CH, Barfoot TD and Sibley G (2015) Continuous-time batch trajectory estimation using temporal basis functions. *International Journal of Robotics Research* 34(14): 1688–1710.
- Gamal AE and Eltoukhy H (2005) CMOS image sensors. *IEEE Circuits and Devices Magazine*.
- Gauglitz S, Foschini L, Turk M and Hollerer T (2011) Efficiently selecting spatially distributed keypoints for visual tracking. In: *18th IEEE International Conference on Image Processing*.
- Golub GH and van Loan CF (1983) *Matrix Computations*. Baltimore, Maryland: Johns Hopkins University Press.
- Guennebaud G, Jacob B et al. (2010) Eigen v3. <http://eigen.tuxfamily.org>.
- Hedborg J, Forssén PE, Felsberg M and Ringaby E (2012) Rolling shutter bundle adjustment. In: *IEEE Conference on Computer Vision and Pattern Recognition*.
- Horn BKP (1987) Solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A*: 629–642.
- Kerl C, Stückler J and Cremers D (2015) Dense continuous-time tracking and mapping with rolling shutter cameras. In: *IEEE*

- International Conference on Computer Vision (ICCV15).*
- Kim JH, Cadena C and Reid I (2016) Direct semi-dense SLAM for rolling shutter cameras. In: *IEEE International Conference on Robotics and Automation (ICRA16)*.
- Kim MJ, Kim MS and Shin SY (1995) A general construction scheme for unit quaternion curves with simple high order derivatives. In: *SIGGRAPH'95*. pp. 369–376.
- Kopf J, Cohen MF and Szeliski R (2014) First-person hyper-lapse videos. In: *ACM Transactions on Graphics (Proc. SIGGRAPH 2014)*.
- Lovegrove S, Patron-Perez A and Sibley G (2013) Spline fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras. In: *British Machine Vision Conference (BMVC)*. BMVA.
- Murray RM, Li Z and Sastry SS (1994) *A Mathematical Introduction to Robotic Manipulation*. CRC Press.
- Oth L, Furgale P, Kneip L and Siegwart R (2013) Rolling shutter camera calibration. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR13)*. Portland, Oregon, pp. 1360–1367.
- Ovrén H (2018) Kontiki - the continuous time toolkit. <https://github.com/hovren/kontiki>.
- Ovrén H and Forssén PE (2015) Gyroscope-based video stabilisation with auto-calibration. In: *IEEE International Conference on Robotics and Automation ICRA'15*.
- Ovrén H and Forssén PE (2018) Spline error weighting for robust visual-inertial fusion. In: *IEEE Conference on Computer Vision and Pattern Recognition*. Salt Lake City, Utah, USA: Computer Vision Foundation.
- Patron-Perez A, Lovegrove S and Sibley G (2015) A spline-based trajectory representation for sensor fusion and rolling shutter cameras. *International Journal on Computer Vision* 113(3): 208–219.
- Rosten E, Porter R and Drummond T (2010) Faster and better: A machine learning approach to corner detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 32(1).
- Shoemake K (1985) Animating rotation with quaternion curves. In: *Int. Conf. on CGIT*. pp. 245–254.
- Strasdat H and Lovegrove S (2011–2017) Sophus: C++ implementation of lie groups using eigen. <https://github.com/strasdat/Sophus>.
- Triggs B, Mclauchlan P, Hartley R and Fitzgibbon A (2000) Bundle adjustment a modern synthesis. In: *Vision Algorithms: Theory and Practice, LNCS*. Springer Verlag, pp. 298–375.
- Unser M, Aldroubi A and Eden M (1993) B-spline signal processing. II. Efficiency design and applications. *IEEE Transactions on Signal Processing* 41(2): 834–848. DOI: 10.1109/78.193221.
- Vedaldi A, Guidi G and Soatto S (2007) Moving forward in structure from motion. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. pp. 1–7. DOI:10.1109/CVPR.2007.383117.
- Young AD, Ling MJ and Arvind DK (2011) IMUSim: A simulation environment for inertial sensing algorithm design and evaluation. *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks* : 199–210.
- Zach C (2014) Robust bundle adjustment revisited. In: *European Conference on Computer Vision ECCV'14*.
- Zefran M and Kumar V (1996) Planning smooth motions on SE(3). In: *Proceedings - IEEE International Conference on Robotics and Automation*, April. pp. 121–126. DOI:10.1109/ROBOT.1996.503583.
- Zhang Z (1997) Parameter estimation techniques: A tutorial with application to conic fitting. *Journal of Image and Vision Computing* 15(1): 59–76.