Linköping Studies in Science and Technology
Dissertations, No. 1951

Continuous Models for Cameras and Inertial Sensors

Hannes Ovrén

Linköping University
Department of Electrical Engineering
Computer Vision Laboratory
SE-581 83 Linköping, Sweden

Linköping 2018

Edition 1:1

ISBN 978-91-7685-244-6
ISSN 0345-7524
URL http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-148766

Typeset using XeTeX

Printed by LiU-Tryck, Linköping 2018

For Mary and Albin

POPULÄRVETENSKAPLIG SAMMANFATTNING

Att använda bilder för att återskapa världen omkring oss i tre dimensioner är ett klassiskt problem inom datorseende. Några exempel på användningsområden är inom navigering och kartering för autonoma system, stadsplanering och specialeffekter för film och spel. En vanlig metod för 3D-rekonstruktion är det som kallas ”struktur från rörelse”. Namnet kommer sig av att man avbildar (fotograferar) en miljö från flera olika platser, till exempel genom att flytta kameran. Det är därför något ironiskt att många struktur-från-rörelse-algoritmer får problem om kameran inte är stilla när bilderna tas, exempelvis genom att använda sig av ett stativ. Anledningen är att en kamera i rörelse ger upphov till störningar i bilden vilket ger sämre bildmätningar, och därmed en sämre 3D-rekonstruktion. Ett välkänt exempel är rörelseoskärpa, medan ett annat är kopplat till användandet av en elektronisk rullande slutare. I en kamera med rullande slutare avbildas inte alla pixlar i bilden samtidigt, utan istället rad för rad. Om kameran rör på sig medan bilden tas uppstår därför störningar i bilden som måste tas om hand om för att få en bra rekonstruktion.

Den här avhandlingen berör robusta metoder för 3D-rekonstruktion med rörliga kameror. En röd tråd inom arbetet är användandet av en tröghetssensor (IMU). En IMU mäter vinkelhastigheter och accelerationer, och dessa mätningar kan användas för att bestämma hur kameran har rört sig över tid. Kunskap om kamerans rörelse ger möjlighet att korrigera för störningar på grund av den rullande slutaren. Ytterligare en fördel med en IMU är att den ger mätningar även i de fall då en kamera inte kan göra det. Exempel på sådana fall är vid extrem rörelseoskärpa, starkt motljus, eller om det saknas struktur i bilden.

Om man vill använda en kamera tillsammans med en IMU så måste dessa kalibreras och synkroniseras: relationen mellan deras respektive koordinatsystem måste bestämmas, och de måste vara överens om vad klockan är. I den här avhandlingen presenteras en metod för att automatiskt kalibrera och synkronisera ett kamera-IMU-system utan krav på exempelvis kalibreringsobjekt eller speciella rörelsemönster.

I klassisk struktur från rörelse representeras kamerans rörelse av att varje bild beskrivs med en kamera-pose. Om man istället representerar kamerarörelsen som en tidskontinuerlig trajektoria kan man på ett naturligt sätt hantera problematiken kring rullande slutare. Det gör det också enkelt att införa tröghetsmätningar från en IMU. En tidskontinuerlig kameratrajektoria kan skapas på flera sätt, men en vanlig metod är att använda sig av så kallade splines. Förmågan hos en spline att representera den faktiska kamerarörelsen beror på hur tätt dess knutar placeras. Den här avhandlingen presenterar en metod för att uppskatta det approximationsfel som uppkommer vid valet av en för gles spline. Det uppskattade approximationsfelet kan sedan användas för att balansera mätningar från kameran och IMU:n när dessa används för sensorfusion. Avhandlingen innehåller också en metod för att bestämma hur tät en spline behöver vara för att ge ett gott resultat.

En annan metod för 3D-rekonstruktion är att använda en kamera som också mäter djup, eller avstånd. Vissa djupkameror, till exempel Microsoft Kinect, har samma problematik med rullande slutare som vanliga kameror. I den här avhandlingen visas hur den rullande slutaren i kombination med olika typer och storlekar av rörelser påverkar den återskapade 3D-modellen. Genom att använda tröghetsmätningar från en IMU kan djupbilderna korrigeras, vilket visar sig ge en bättre 3D-modell.

ABSTRACT

Using images to reconstruct the world in three dimensions is a classical computer vision task. Some examples of applications where this is useful are autonomous mapping and navigation, urban planning, and special effects in movies. One common approach to 3D reconstruction is ”structure from motion” where a scene is imaged multiple times from different positions, e.g. by moving the camera. However, in a twist of irony, many structure from motion methods work best when the camera is stationary while the image is captured. This is because the motion of the camera can cause distortions in the image that lead to worse image measurements, and thus a worse reconstruction. One such distortion common to all cameras is motion blur, while another is connected to the use of an electronic rolling shutter. Instead of capturing all pixels of the image at once, a camera with a rolling shutter captures the image row by row. If the camera is moving while the image is captured the rolling shutter causes non-rigid distortions in the image that, unless handled, can severely impact the reconstruction quality.

This thesis studies methods to robustly perform 3D reconstruction in the case of a moving camera. To do so, the proposed methods make use of an inertial measurement unit (IMU). The IMU measures the angular velocities and linear accelerations of the camera, and these can be used to estimate the trajectory of the camera over time. Knowledge of the camera motion can then be used to correct for the distortions caused by the rolling shutter. Another benefit of an IMU is that it can provide measurements also in situations when a camera can not, e.g. because of excessive motion blur, or absence of scene structure.

To use a camera together with an IMU, the camera-IMU system must be jointly calibrated. The relationship between their respective coordinate frames need to be established, and their timings need to be synchronized. This thesis shows how to automatically perform this calibration and synchronization, without requiring e.g. calibration objects or special motion patterns.

In standard structure from motion, the camera trajectory is modeled as discrete poses, with one pose per image. Switching instead to a formulation with a continuous-time camera trajectory provides a natural way to handle rolling shutter distortions, and also to incorporate inertial measurements. To model the continuous-time trajectory, many authors have used splines. The ability for a spline-based trajectory to model the real motion depends on the density of its spline knots. Choosing a too smooth spline results in approximation errors. This thesis proposes a method to estimate the spline approximation error, and use it to better balance camera and IMU measurements, when used in a sensor fusion framework. Also proposed is a way to automatically decide how dense the spline needs to be to achieve a good reconstruction.

Another approach to reconstruct a 3D scene is to use a camera that directly measures depth. Some depth cameras, like the well-known Microsoft Kinect, are susceptible to the same rolling shutter effects as normal cameras. This thesis quantifies the effect of the rolling shutter distortion on 3D reconstruction, depending on the amount of motion. It is also shown that a better 3D model is obtained if the depth images are corrected using inertial measurements.

Acknowledgments

When I applied to Linköping University in 2004, my father challenged me in my choice to study computer engineering, because ”computers are just tools”. I did not necessarily agree with that, but at least it made me question whether I actually wanted to build a career on computers, or if I just liked computer games. With this thesis, maybe I can finally put that question to rest.

My years as a PhD student have probably been the greatest time of my life (so far). I have worked on interesting projects, seen new places, and met many interesting people. I would like to thank everyone at CVL, both past and present, for sharing their knowledge, and for their good company at coffee breaks.

Some people have had a greater impact on my time as a PhD student, or the writing of this thesis:

- My supervisor, **Per-Erik Forssén**, has been an invaluable source of both scientific advice, and moral support.
- My co-supervisors **Klas Nordberg** and **David Törnqvist**, who provided valuable insights into geometry and control theory, respectively.
- **Andreas Robinson** designed our latest IMU hardware, which made data capture so much more convenient.
- **Jörgen Ahlberg**, **Amanda Berg**, **Emil Brissman**, **Per-Erik Forssén**, **Bertil Grelsson**, **Mary Hagelin Emilsson**, **Gustav Häger**, **Klas Nordberg**, and **Joakim Rydell**, proof read parts of this manuscript, for which I am very grateful.
- **Martin Danelljan** kindly shared his thesis chapter styles.
- I shared offices with **Marcus Wallenberg** and **Giulia Meneghetti**, who are both excellent, but very different people. Thanks for all the great discussions!

A happy researcher is an efficient researcher. To keep me sane, and my spirits up, the following people deserves special credit:

- **My family**, for always believing in me, and supporting me in all my endeavors.
- All my dear friends: **Daniel**, **Mika**, **Erika**, **Tommaso**, **Giulia**, **Kristoffer**, **Joakim**, and everyone else who are not on this list, but are definitely not forgotten.
- My exercise buddies at **Apstark**, who have helped me keep in shape, and made me (re)discover how fun it is to climb things.
- **Mary**, your love and support is what kept me going when times were tough. I could not have done this without you, and I look forward to all our future adventures!
- **Albin**, this thesis regretfully took most of my attention during your first weeks in this world, but I hope to make up for that in the years to come.

This work was supported by the Swedish Research Council through grants for the projects EVOR (2008-4509), LCMM (2014-5928), and EMC2 (2014-6227); by the Swedish Foundation for Strategic Research through a grant for the project VPS (IIS11-0081); and by Linköping University.

Linköping, July 2018
Hannes Ovrén

About the cover

The cover shows one video frame from the *Handheld 2* dataset from Paper C, recorded in the apple garden at Campus Valla, Linköping University. Blended with this video frame is the 3D structure that resulted after solving a continuous-time structure from motion problem, using the video and inertial data. The red curve is the continuous-time camera trajectory, as seen from the current frame. A big thank you to Tomas Hägg at LiU-Tryck for refining my first draft of the cover.

Contents

Part I

Background

1

Introduction

1.1 Motivation

Recreating the world in three dimensions using images is an old idea that dates back hundreds of years. More recent improvements in the form of digital cameras, and powerful computers, have however made this idea more feasible than ever. In the field of computer vision, the 3D reconstruction problem is usually referred to as *structure from motion*, because it requires images taken from different positions, i.e. as from a moving camera. Today there are companies that use structure from motion to build 3D models of everything from small everyday objects, to buildings, or even entire cities and planets.

Another concept which is quickly gaining in popularity is *augmented reality* (AR). Both Apple and Google just recently released new versions of AR development tools for their respective mobile operating systems. Using either a screen (e.g. a smartphone or tablet) or a head-up display, the idea of AR is to add contextual information to the observer's view. For example, a smartphone app can show information about an item in a museum exhibit, or a famous historical landmark. In an industry setting, AR can be used to provide a worker with assembly instructions, which are overlaid on her field of view, using a head-up display. To put the information in the right place on the screen, the AR application needs to know both where the camera is, and where objects in the scene are, at all times. This is the definition of the *simultaneous localization and mapping* (SLAM) problem, which has been studied in the field of robotics, and of which structure from motion is a special case. In SLAM it is common to combine the visual measurements from the camera with inertial measurements from an *inertial measurement unit* (IMU). This allows for more robust camera pose estimates, but also allows finding the true scale of the scene, which is otherwise not possible using a single camera. Having a scene with correct scale can be important in AR applications, e.g. to allow placing virtual furniture in a room. IMUs are available in all smart-

phones, and also some consumer cameras (like newer versions of the GoPro Hero series), which makes adding inertial measurements a possibility for 3D reconstruction in many cases.

One problem with using structure from motion (or visual SLAM in general) is that virtually all consumer cameras, including over two billion smartphones, have what is known as an electronic *rolling shutter*. This rolling shutter breaks a key assumption used in most structure from motion methods, which is that all measurements in a specific image are taken from the same viewpoint, or pose. In the rolling shutter camera, different rows instead have different viewpoints, if the camera is moving. Because of this, most structure from motion methods either require a camera with a global shutter (which are more rare), or that the camera is standing still at all times when images are taken (which is ironic considering the name of the method).

To solve the issue with rolling shutter, one approach is to model the camera pose trajectory as a continuous-time function, instead of using only one pose per image. As an added bonus, the continuous-time formulation also allows for a natural way to incorporate the inertial measurements, by differentiating the trajectory.

1.2 Contributions

One of the most common representations of continuous-time trajectories, is to model them as splines. However, if this spline is not dense enough, it will only approximate the true camera motion. This approximation introduces errors in the least-squares solution to the structure from motion problem. Paper C presents a method to model this approximation, which can be used to correctly balance visual and inertial measurements, and also to set an appropriate spline density. This makes the structure from motion solution more robust, as it allows for a wider range of motions, including using body-mounted cameras, and cameras mounted on a vehicle driving in rough terrain.

The splined trajectories can be constructed in different spaces, where $\mathbb{SE}(3)$ and a combination of $\mathbb{R}^3$ and $\mathbb{SO}(3)$, are the most common choices. Where researchers have previously committed to one of these spaces, Paper D compares them, both empirically and theoretically. The same paper also compares different ways to project 3D points into a rolling shutter camera.

To combine the visual measurements from the camera, with the inertial measurements from the IMU, the camera-IMU system needs to be calibrated and synchronized. Paper B proposes a method for calibration and synchronization that can be performed *after* the data has been recorded, and thus does not require any special calibration equipment or procedure. This method also addresses the problem of sensors that are related by an unknown time scaling factor.

3D reconstruction can be made simpler by using a camera that also directly measures the depth associated with each pixel. Many of these depth cameras are equipped with a rolling shutter, and Paper A investigates how this affects the 3D reconstruction, for different types of motions. A method to correct the depth images using an IMU is also presented, which results in better 3D models.

1.3 Outline

This thesis is divided into two parts, where part I contains background theory, and an overview of the contributions. Part II contains the publications which are included in this thesis.

In part I, chapter 2 describes the different sensors (cameras and IMUs), and how their measurements are formed. Chapter 3 gives an overview of the area of continuous-time structure from motion, and chapter 4 looks at the specific case of using splines as trajectories. Sensor calibration is discussed in chapter 5, and methods for evaluation in chapter 6. Finally, chapter 7 gives a summary, and looks towards the future.

1.4 Included publications

Paper A: "Improving RGB-D Scene Reconstruction Using Rolling Shutter Rectification"

> Hannes Ovrén, Per-Erik Forssén, and David Törnqvist. "Improving RGB-D Scene Reconstruction Using Rolling Shutter Rectification". In: *New Development in Robot Vision*. 2014, pp. 55–71. ISBN: 978-3-662-43858-9. DOI: 10.1007/978-3-662-43859-6_4
> **Invited paper, extended from our submission to Workshop on Robot Vision 2013.**

Abstract: Scene reconstruction, i.e. the process of creating a 3D representation (mesh) of some real world scene, has recently become easier with the advent of cheap RGB-D sensors (e.g. the Microsoft Kinect).
Many such sensors use rolling shutter cameras, which produce geometrically distorted images when they are moving. To mitigate these *rolling shutter distortions* we propose a method that uses an attached gyroscope to rectify the depth scans. We also present a simple scheme to calibrate the relative pose and time synchronization between the gyro and a rolling shutter RGB-D sensor.
For scene reconstruction we use the Kinect Fusion algorithm to produce meshes. We create meshes from both raw and rectified depth scans, and these are then compared to a ground truth mesh. The types of motion we investigate are: pan, tilt and wobble (shaking) motions.
As our method relies on gyroscope readings, the amount of computations required is negligible compared to the cost of running Kinect Fusion.
This chapter is an extension of a paper at the IEEE Workshop on Robot Vision. Compared to that paper, we have improved the rectification to also correct for lens distortion, and use a coarse-to-fine search to find the time shift more quickly. We have extended our experiments to also investigate the effects of lens distortion, and to use more accurate ground truth. The experiments demonstrate that correction of rolling shutter effects yields a larger improvement of the 3D model than correction for lens distortion.
Author's contributions: The author wrote software for data collection and experimental evaluation, performed calibration of the system, and contributed to designing the calibration and synchronization method and experiments. The author also contributed to writing the paper.

Paper B: "Gyroscope-based video stabilisation with auto-calibration"

Hannes Ovrén and Per-Erik Forssén. "Gyroscope-based video stabilisation with auto-calibration". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, WA: IEEE, May 2015, pp. 2090–2097. ISBN: 978-1-4799-6923-4. DOI: 10.1109/ICRA.2015.7139474

Abstract: We propose a technique for joint calibration of a wide-angle rolling shutter camera (e.g. a GoPro) and an externally mounted gyroscope. The calibrated parameters are time scaling and offset, relative pose between gyroscope and camera, and gyroscope bias. The parameters are found using non-linear least squares minimisation using the symmetric transfer error as cost function.

The primary contribution is methods for robust initialisation of the relative pose and time offset, which are essential for convergence. We also introduce a robust error norm to handle outliers. This results in a technique that works with general video content and does not require any specific setup or calibration patterns.

We apply our method to stabilisation of videos recorded by a rolling shutter camera, with a rigidly attached gyroscope. After recording, the gyroscope and camera are jointly calibrated using the recorded video itself. The recorded video can then be stabilised using the calibrated parameters.

We evaluate the technique on video sequences with varying difficulty and motion frequency content. The experiments demonstrate that our method can be used to produce high quality stabilised videos even under difficult conditions, and that the proposed initialisation is shown to end up within the basin of attraction. We also show that a residual based on the symmetric transfer error is more accurate than residuals based on the recently proposed epipolar plane normal coplanarity constraint, and that the use of robust errors is a critical component to obtain an accurate calibration.

Author's contributions: The author built hardware for data collection and wrote software for data collection and experiments, and contributed to the design of the auto-calibration method and experiments on robustness and shape of the basin of convergence. Apart from also contributing to writing the paper, the author packaged and released the calibration method under an open source license.

Paper C: "Spline Error Weighting for Robust Visual-Inertial Fusion"

> Hannes Ovrén and Per-Erik Forssén. "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, June 2018

Abstract: In this paper we derive and test a probability-based weighting that can balance residuals of different types in spline fitting. In contrast to previous formulations, the proposed spline error weighting scheme also incorporates a prediction of the approximation error of the spline fit. We demonstrate the effectiveness of the prediction in a synthetic experiment, and apply it to visual-inertial fusion on rolling shutter cameras. This results in a method that can estimate 3D structure with metric scale on generic first-person videos. We also propose a quality measure for spline fitting, that can be used to automatically select the knot spacing. Experiments verify that the obtained trajectory quality corresponds well with the requested quality. Finally, by linearly scaling the weights, we show that the proposed spline error weighting minimizes the estimation errors on real sequences, in terms of scale and end-point errors.

Author's contributions: The author wrote the software for continuous-time structure from motion, data collection, and experiments. Contributed to the residual weighting scheme, and knot selection method. The author also contributed to writing the paper.

Paper D: "Trajectory Representation and Landmark Projection for Continuous-Time Structure from Motion"

> Hannes Ovrén and Per-Erik Forssén. "Trajectory Representation and Landmark Projection for Continuous-Time Structure from Motion". In: *International Journal of Robotics Research* XXX.XXX (2018). Under review

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 $\mathcal{C}^2$-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.

Author's contributions: The author wrote the software for continuous-time structure from motion, data collection, and experiments. Apart from contributing to writing the paper, the author also contributed to the theoretical analysis of the tested reprojection methods, and to the experiment design. The author also packaged and released the continuous-time structure from motion framework under an open source license.

1.5 Other publications

The following publications by the author are related to the included papers.

Peer-reviewed

- Hannes Ovrén, Per-Erik Forssén, and David Törnqvist. "Why Would I Want a Gyroscope on my RGB-D Sensor?" In: *Proceedings of IEEE Winter Vision Meetings, Workshop on Robot Vision (WoRV13)*. Clearwater, FL, USA: IEEE, Jan. 2013
  (Paper A is an extension of this work.)

- Felix Järemo Lawin, Per-Erik Forssén, and Hannes Ovrén. "Efficient Multi-Frequency Phase Unwrapping using Kernel Density Estimation". In: *European Conference on Computer Vision (ECCV)*. Amsterdam: Springer International Publishing AG, Oct. 2016
  (Supervised the Master's thesis on which this work is based. Assisted with experiments.)

Non-reviewed symposium submissions

- Hannes Ovrén, Per-Erik Forssén, and David Törnqvist. "Better 3D with Gyroscopes". In: *Proceedings of SSBA 2013 IAPR*. Non-reviewed workshop. IAPR. SSBA, Mar. 2013
  (Shortened version of the WoRV'13 submission)

- Hannes Ovrén and Per-Erik Forssén. "Camera-IMU Calibration with Robust Initialisation". In: *Proceedings of SSBA 2015 IAPR*. Non-reviewed workshop. IAPR. SSBA, Mar. 2015
  (Shortened version of Paper B)

- Hannes Ovrén and Per-Erik Forssén. "Ground Truth for Rolling Shutter Visual-Inertial SLAM and Camera-IMU Calibration". In: *Proceedings of SSBA 2016 IAPR*. Non-reviewed workshop. IAPR. SSBA, Mar. 2016
  (Describes a way to generate realistic synthetic ground truth)

- Hannes Ovrén and Per-Erik Forssén. "Inertial-aided Continuous-Time Structure from Motion in Practice". In: *Proceedings of SSBA 2018 IAPR*. Non-reviewed workshop. IAPR. SSBA, Mar. 2018
  (Shortened version of Paper C)

2

Sensors

Fusion of camera and inertial measurements is quite common because they have complementary characteristics. By observing a 3D point in multiple images, the relative orientation between the image viewpoints can be estimated with high accuracy, and without bias. The measurements from an inertial measurement unit (IMU), are distorted by biases, which cause drift in estimates of the orientation and position. The IMU can, however, provide measurements also in situations when the camera can not, e.g. because of excessive motion blur, or lack of structure in the scene. An IMU typically also has a sample rate that is significantly higher than the camera frame rate.

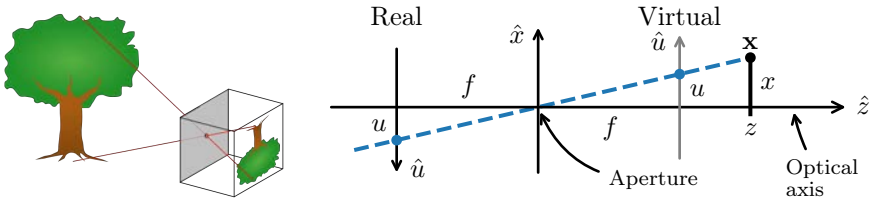
This chapter gives an overview of cameras and inertial sensors, and explains how their measurements are formed.

2.1 Cameras

A camera is a device that captures the light which is reflected or emitted from objects in the world, and records it as an *image*. In the analog days, the image was captured on film with a coating that reacts to the rays of light hitting it. Today, images are typically digital, and are captured by a grid of light-sensitive elements on a sensor chip. The end product is a digital image which consists of a (usually) uniform grid of *picture elements*, or *pixels*, that represent the intensity and color of the captured light.

The pinhole camera

To use measurements gathered by a camera, we must have a mathematical model of how an image is formed. Specifically, we need to know how a 3D point $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^T$, expressed in the camera coordinate frame, is projected to its image plane location $\mathbf{y} = \begin{bmatrix} u & v \end{bmatrix}^T$. While 3D points are usually expressed in a world coordinate frame, to allow for more than one camera, the projections



(a) (b)

Figure 2.1: The pinhole camera. (a) shows how a 3D scene is projected into an image, formed by the aperture in the pinhole camera. (b) shows the geometry of how a 3D point, $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^T$ (here shown only on the xz-plane), is projected onto the image u-axis, given the focal length, f. Both the real and virtual image planes are shown.

described in this chapter are always in the camera coordinate frame. The case with multiple cameras is instead deferred to chapter 3.

The most prevalent model in computer vision is that of the *pinhole camera*, which is illustrated in figure 2.1a. Mathematically, a pinhole camera consists of an *image plane* where the image is formed, and a tiny hole, the *aperture*, that forms it. The distance between the aperture and the image plane, along the *optical axis*, is known as the *focal length*, f.

In figure 2.1a we can see that the image formed on the image plane is upside down. To avoid this, we can instead imagine a *virtual image plane* placed in front of the aperture. While the virtual image plane is not possible in practice, it allows for a nicer mathematical formulation. In the pinhole camera model, the projection is specified by the focal length, which is illustrated in figure 2.1b. By similarity of triangles, we get the expression

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f\frac{x}{z} \\ f\frac{y}{z} \end{bmatrix}. \tag{2.1}$$

The pinhole projection can also be written in matrix form, using homogeneous coordinates:

$$\mathbf{y} = \begin{bmatrix} u \\ v \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{K}\mathbf{x}. \tag{2.2}$$

Here $\sim$ denotes equality after projection of the right operand, i.e.

$$\mathbf{y} \sim \mathbf{x} \quad \Leftrightarrow \quad \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x/z \\ y/z \end{bmatrix}. \tag{2.3}$$

The matrix $\mathbf{K}$ is called the *intrinsic matrix*. In (2.2) it only contains the focal length, but in general it could be any upper triangular matrix, where

one possible factorization is

$$\mathbf{K} = \begin{bmatrix} sf & \gamma & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.4}$$

The s parameter takes care of the case where the scaling of the horizontal and vertical axis are not equal, while γ models shearing. s and γ makes it possible to handle slight manufacturing errors that leave the sensor chip not exactly aligned with the optical axis, although one should be careful to not apply too much physical interpretation of these values. u_0 and v_0 are offsets that moves the image origin to the top-left corner, instead of the center of the image plane. Having the origin in the top-left corner, with the v-axis pointing downwards, is more convenient in a digital image representation.

Lens distortion

The problem with pinhole cameras is that they need a small aperture to get a sharp image. This either results in very little light hitting the sensor chip, and a very dark image, or long exposure times. Long exposure times are generally unwanted because they cause *motion blur* when either the camera, or objects in the scene, are moving during the exposure. To overcome this, a camera is usually equipped with an optical system that contains one or more lenses that can collect more light, while still providing a sharp image.

Instead of using physics to model the exact system of lenses used by the camera, computer vision researchers tend to instead favour simplified *lens distortion models*. While these might not be able to exactly model the optical system, a correctly chosen lens model is usually sufficient for most applications.

Lens distortion can be modelled in different ways, but the representation chosen here is as a distortion

$$\mathbf{y}_d = \phi(\mathbf{y}_n), \tag{2.5}$$

where $\mathbf{y}_n \sim \mathbf{x}$ are the image coordinates of the projected 3D point on the image plane of a normalized camera ($\mathbf{K} = \mathbf{I}$). The image plane distortion is then followed by a pinhole projection

$$\mathbf{y} \sim \mathbf{K} \begin{bmatrix} \mathbf{y}_d \\ 1 \end{bmatrix}. \tag{2.6}$$

The lens distortion function $\phi(\mathbf{y}_n)$ can usually be decomposed into two parts: one radial and one tangential. The radial part models the distortion along a line from the center of distortion to the normalized image point $\mathbf{y}_n$, while the tangential part is a distortion orthogonal to this direction.

The choice of lens model depends on the characteristics of the camera. The camera used in Paper B, Paper C, and Paper D, is an action camera, with a horizontal field of view of approximately 120°. A suitable model which can account for the wide-angle characteristics of this type of camera is the FOV lens model by Devernay and Faugeras [12]. With the addition of a distortion center, and a simplification [28], this radial lens distortion model is defined as

$$\mathbf{y}_d = \phi(\mathbf{y}_n) = \mathbf{d} + \frac{\arctan(r\lambda)}{\lambda} \cdot \frac{(\mathbf{y}_n - \mathbf{d})}{r}, \quad \text{where} \tag{2.7}$$

$$r = \|\mathbf{y}_n - \mathbf{d}\|. \tag{2.8}$$

Here, λ is a distortion parameter, and $\mathbf{d} = \begin{bmatrix} d_x & d_y \end{bmatrix}^T$ is the distortion center. The inverse model[1], which produces the corresponding normalized image coordinate is similarly defined as

$$\mathbf{y_n} = \phi^{-1}(\mathbf{y}_d) = \mathbf{d} + \frac{\tan(r\lambda)}{\lambda} \cdot \frac{\mathbf{y}_d - \mathbf{d}}{r}, \quad \text{where} \tag{2.9}$$

$$r = \|\mathbf{y}_d - \mathbf{d}\|, \tag{2.10}$$

$$\mathbf{y}_d \sim \mathbf{K}^{-1} \begin{bmatrix} \mathbf{y} \\ 1 \end{bmatrix}. \tag{2.11}$$

Since the FOV lens model only has three degrees of freedom (λ, d_x, and d_y), it is simple and robust to estimate. Camera calibration is discussed in chapter 5.

The projection function, $\pi(\cdot)$

When the exact details of the camera projection are not important, the projection can be summarized using the projection function $\pi(\cdot)$, defined as

$$\mathbf{y} = \pi(\mathbf{x}) \sim \mathbf{K} \begin{bmatrix} \phi(\mathbf{y}_n) \\ 1 \end{bmatrix}, \tag{2.12}$$

where $\mathbf{y}_n \sim \mathbf{x}$. This function includes both the lens distortion model and intrinsic matrix.

Since the distance to the 3D point is lost in the projection, the inverse of the projection function can only reconstruct the 3D point up to scale. Here it is defined as

$$\pi^{-1}(\mathbf{y}) = \begin{bmatrix} \phi^{-1}(\mathbf{y}_d) \\ 1 \end{bmatrix}, \tag{2.13}$$

where $\mathbf{y}_d$ is defined in (2.11). The true 3D point can be recovered only if its depth (i.e. z coordinate) is known:

$$\mathbf{x} = z\pi^{-1}(\mathbf{y}). \tag{2.14}$$

[1] The inverse model is an approximation which assumes that $\mathbf{d}$ is small.

Rolling shutter

With an analog camera, the image is formed as light reacts with the surface of the photographic film. This reaction depends on the amount of light that hits the film, which can be controlled by a shutter. The shutter is a screen which either blocks the light, or lets it pass, and is designed to switch between these states very quickly. In low-light conditions, the photographer increases the time which the shutter stays open, allowing more light to hit the film.

Where the analog camera has a photosensitive film, the digital camera has a digital imaging sensor. The imaging sensor consists of a large number of photo diodes, which react when hit by light. Analog to digital converters are then used to convert the analog values (voltages or charges) of the photo diodes to digital values. Early digital cameras typically used an image sensor based on a technology called *charge coupled device* (CCD), but in recent years the market has instead been dominated by cameras based on *complementary metal-oxide semiconductors* (CMOS). CCDs can still be found mainly in industrial applications, whereas virtually all consumer cameras (including smartphones) are based on CMOS. The main reasons why CMOS has become dominant are because it is cheaper to manufacture, and also allows for higher capture rates [21].

To capture an image, light is allowed to fall on the sensor chip, and each photo diode registers the amount of light that reacts with it, by increasing a charge level or voltage. The *integration time* is the time during which the photo diodes are allowed to collect the light. A dark scene requires more light than a bright one, which results in an increased integration time.

The main difference between CCD and CMOS is in how they read out the charges or voltages from the photo diodes. In a CCD-based sensor chip, the charge stored in the photo diode is transferred into a CCD. Since all photo diodes have their own CCD, this can be done for all photo diodes at once. After the charges have been transferred to the CCDs, they can be read out sequentially. In contrast, a CMOS sensor operates in a manner similar to random access memory, by selecting the active row and then reading out all pixels of the row at once. Since CMOS lacks the intermediate charge storage of the CCD-sensor, it must not only read out the rows one by one, but also integrate the light row-by-row. If it did not, then the last rows of the sensor would get more light than the first rows, which is obviously not desirable. In practice, this means that the pixels captured by a CCD camera are all integrated over the same period of time, and we therefore say that it has a *global shutter*. In the CMOS sensor, each row of the resulting image is instead integrated over different periods of time, and because of this we say that it has a *rolling shutter* [2]. Figure 2.2 shows a timing schedule for integration and readout for both global and rolling shutters.

[2] There are examples of CMOS sensors with a global shutter, but these are not common.

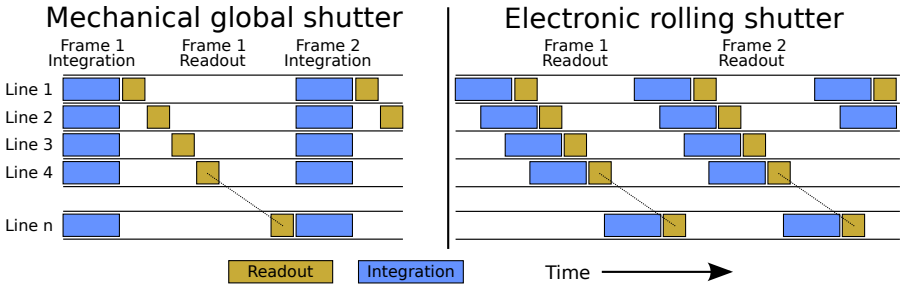


Figure 2.2: Timing schedule for global and rolling shutter cameras. The x-axis is time, and the y-axis is image row number. The timing for each row is divided into integration time and row readout. Image from [59], used with permission.

We can model the rolling shutter by its *image readout time*, r, which is defined as the time it takes to integrate and read out all N image rows. This means that the pixels of row v are captured at time

$$t_v = t_0 + v\frac{r}{N}\,, \tag{2.15}$$

where t_0 is the time of the first row. With a global shutter camera, the capture time for all pixels is simply t_0, the start-of-frame time.

If the scene is static and the camera is stationary, an image captured by a rolling shutter camera will be identical to one captured by a global shutter camera. However, if the camera, or objects in the scene, are moving, then the rolling shutter camera image will be distorted. An example of this is shown in figure 2.3.

Tracking and matching image features

The previous sections explain how an image measurement, $\mathbf{y}$, corresponds to a 3D point, $\mathbf{x}$, by camera projection. But how are these image measurements tracked over time?

The input to the 3D reconstruction problem is a set of *feature tracks* $\mathcal{Y} = \{\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_K\}$. The feature track for a 3D point $\mathbf{x}_k$ is defined as the set of image measurements

$$\mathcal{Y}_k = \{\mathbf{y}_{k,i} : i \in \mathcal{I}_k\}\,, \tag{2.16}$$

where $\mathbf{y}_{k,i}$ is the projection of $\mathbf{x}_k$ in image i, and $\mathcal{I}_k$ is the set of images where $\mathbf{x}_k$ is visible[3].

A general method to find feature tracks, given unordered images, is to

[3] $\mathbf{x}_k$ is now a point in the world coordinate frame, which is moved into the camera coordinate frame as defined in chapter 3

(a) Original (with rolling shutter) (b) Rectified (no rolling shutter)

Figure 2.3: Example of rolling shutter distortion caused by a moving camera. The rolling shutter effect in the original image is most apparent by looking at the pole, which appears bent. On the right is a rectified version of the original, where the rolling shutter distortion has been removed. Images taken from [18], used with permission.

1. detect *keypoints* in all images $\mathcal{I}$ using a *feature detector*,
2. compute a *feature descriptor* for each keypoint, and
3. group keypoints with similar descriptors, using feature matching, to form a track.

The role of the feature detector is to find features (locations) in the image which have a high probability of being redetectable in other images. For example, a point on a blank wall is hard to reliably redetect since the lack of structure makes it difficult to find the exact same spot. A point located at the corner of a door frame is however very likely to be redetectable in many different images because its appearance is likely to be locally unique in each image.

To match keypoints to each other, a feature descriptor, $\mathbf{f}$, is computed using the image neighbourhood of the keypoint. A matching function $m(\mathbf{f}_a, \mathbf{f}_b)$ defines a metric that indicates how similar two descriptors are, and this can be used to group the keypoints into tracks. The most well-known example of a feature detector and descriptor is SIFT by Lowe [41].

In the case of video data, the images are temporally ordered, and it can be assumed that the camera has not moved very far between one image and the next one. In this case the detect-compute-match procedure can be simplified to instead use *feature tracking*. After detecting keypoints in image n, the corresponding keypoints in image $n+1$ are found directly by tracking each keypoint to the new image. This thesis uses the LK tracker by Lucas and Kanade [42] which finds the new keypoint location by minimizing the difference between patches around the source keypoint and the new keypoint. The

LK-tracker can be used with any keypoint detector, and is in this thesis used together with *Good Features to Track* by Shi and Tomasi [64], or FAST by Rosten et al. [62].

Back tracking

Using tracking instead of feature matching improves speed since only features between two images need to be matched, instead of between all images. Tracking can however cause drift if the features are tracked over a long sequence. The drift can be somewhat mitigated by also tracking keypoints backwards and making sure that the backward and forward tracks are identical [26]. This does however require more computations, and more memory, since a list of recent images need to be stored to perform the backwards tracking. For global shutter cameras, the quality of the feature tracks can be improved by enforcing geometric consistency, which is described in section 3.8.

2.2 Depth cameras

Instead of only capturing light intensity, there are cameras that for each pixel also provide a measure of the distance to the imaged object. These sensors are commonly known as depth cameras.

The depth camera provides a *depth image*, $\mathbf{D}(\mathbf{y})$, that maps every image point $\mathbf{y}$ to a depth value. The depth value is simply the z coordinate of the corresponding 3D point, in the camera coordinate frame. The depth image can then be used to, for each pixel, directly compute the corresponding 3D point

$$\mathbf{x}(\mathbf{y}) = \mathbf{D}(\mathbf{y})\pi^{-1}(\mathbf{y})\,. \tag{2.17}$$

Many depth cameras contain also a regular, visual, camera. These are typically referred to as *RGB-D* cameras, because they capture both color (RGB) and depth (D). With an RGB-D camera it is possible to get colored 3D data at video rates, which is e.g. used to construct high-quality models in real-time [48], and for robot navigation [34, 33].

Stereo cameras

One way to measure depth is to imitate the human optical system and use two cameras to compute depth from disparity. By finding corresponding points in the two images the depth can be computed if the relative pose (orientation and translation) between the two cameras is known. The same geometry which is involved in stereo vision is used in chapter 3 in the context of structure from motion.

Structured light sensors

One RGB-D sensor which has found both commercial and scientific success is the first *Microsoft Kinect*, which was sold as an add-on to Microsoft's gaming console Xbox 360. It allows a person playing a game to use the body as a controller, by employing skeleton tracking [66], using the RGB-D data as input. While the success of the Kinect among gamers can be debated, it quickly became popular in the scientific community, who now had access to an inexpensive 3D sensor.

The Kinect employs a technique called *structured light sensing*, or *structured light projection* (SLP), to estimate depth [72]. An SLP sensor consists of two components: a camera, and a projector. The projector illuminates the scene with a known pattern of light (the Kinect uses randomly distributed dots). The projected pattern is visible in the camera image, and, since the pattern is known, it is possible to create a correspondence map between the camera image, and the pattern. Computation of the depth is then done as in stereo vision, except that one of the cameras are replaced by a "virtual camera" in the form of the projector. While SLP cameras can use the visual light spectrum, it is more common to instead use an infrared (IR) projector and camera. Working in IR reduces the risk of the structured light pattern becoming visible in the RGB-camera.

If the IR camera is made using CMOS, it has the same problem with rolling shutter as any other camera. In Paper A it is shown that the rolling shutter present on the Kinect causes measurable 3D model errors even for relatively modest camera motions. Also shown is that rectification of the depth images using an IMU (specifically a gyroscope), can remove most of these errors.

Time of flight sensors

At its core, a *time of flight* (ToF) sensor works by emitting a known signal, which is reflected in the scene, to then return back to the sensor. By measuring the time between emitting and receiving the signal (i.e. the time of flight), the distance to the object can be computed using the known speed of light, c. The second version of the Kinect, released for the Xbox One gaming console, used ToF instead of SLP, which provided an upgrade in maximum range and depth image quality [63]. It also switched from rolling to near global shutter, which makes it less interesting for the work presented in this thesis. In [38], to which the author made contributions, a method that increased the depth map quality of the Kinect for Xbox One is presented.

2.3 Inertial measurement units

An inertial measurement unit, or IMU, is a device that can sense the motion of the body which it is attached to. Specifically it measures angular velocity

and linear acceleration, using a *gyroscope*, and an *accelerometer*, respectively. Some IMUs also contain a *magnetometer* which senses the Earth's magnetic field to give an absolute heading, but this was not used in this thesis. By integrating the angular velocity from the gyroscope, and the linear accelerations from the accelerometer, the orientation and position of the sensor, relative to an initial state, can be estimated. IMUs have been used for aerial and naval navigation for half a century, and in the case of gyroscopes even longer. Today, all smartphones contain at least an accelerometer to detect screen orientation changes, while many also feature a gyroscope and/or magnetometer e.g. for games and navigation [13].

Gyroscopes

Imagine an object that is moving at constant velocity relative to a rotating coordinate frame. To an outside observer, the object trajectory is no longer straight, but curved, because of the rotation of the coordinate frame [13]. This implies that a force must be acting on the object, and this *Coriolis force* is what is measured by a gyroscope. To obtain the complete 3D angular velocity vector, three gyroscopes are aligned such that their axes of rotation are orthogonal.

The ideal output of a gyroscope is the angular velocity

$$\boldsymbol{\omega}_0 = \mathbf{R}\boldsymbol{\omega}_w \,, \tag{2.18}$$

where $\boldsymbol{\omega}_w = \begin{bmatrix}\omega_x & \omega_y & \omega_z\end{bmatrix}^T$ is the angular velocity of the sensor, expressed in the world (inertial) coordinate frame, and $\mathbf{R}$ is the rotation between the world and body coordinate frames. In practice, this measurement is disturbed by a number of error sources, where the most important ones are bias and measurement noise. The bias is an offset from the true value which usually varies slowly over time, and needs to be estimated every time the gyroscope is used. The measurement noise is usually modelled as a zero-mean Gaussian random variable. Other sources of errors include e.g. quantization in the analog-to-digital-conversion, non-orthogonality of the measurement axes, and output value scaling.

The model used in this thesis accounts only for the bias, $\mathbf{b}_g$, and measurement noise, $\mathbf{n}_g$:

$$\boldsymbol{\omega} = \boldsymbol{\omega}_0 + \mathbf{b}_g + \mathbf{n}_g \,. \tag{2.19}$$

Accelerometers

A mechanical accelerometer is in essence a mass-spring-dampener-system where a force applied to the mass extends or contracts the spring. This mass displacement, which can be measured, is proportional to the magnitude of the force. Electronic accelerometers work similarly but replace the spring

and dampener with an electronic sensor. In a piezoelectric accelerometer, the mass is placed on top of a stress-sensitive crystal which generates a voltage depending on the amount of force [13]. A capacitive accelerometer instead works by measuring the difference in capacitivity for a set of electrodes as the force moves the mass.

An accelerometer does not only measure the acceleration caused by the motion of the sensor, but also the Earth's gravity. This results in the ideal measurement model

$$\mathbf{a}_0 = \mathbf{R}(\mathbf{a}_w + \mathbf{g}_w)\,, \tag{2.20}$$

where $\mathbf{a}_w = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T$ and $\mathbf{g}_w$ are the linear acceleration of the sensor, and Earth's gravity, respectively, given in the world coordinate frame. $\mathbf{R}$ is the rotation between the world and body coordinate frame. In some applications the inclusion of gravity is useful, as it gives information about which direction is "up". For example, a smartphone or tablet with only an accelerometer can use the direction of the up vector to correctly rotate the screen contents. In navigation applications where the position is estimated by integrating the linear acceleration $\mathbf{a}_w$, the gravitational component must be subtracted before the integration. This is difficult since it requires accurate knowledge of the orientation, $\mathbf{R}$, for every accelerometer sample to integrate.

The same sources of errors for the gyroscope apply also for the accelerometer. The accelerometer model used in this thesis is thus

$$\mathbf{a} = \mathbf{a}_0 + \mathbf{b}_a + \mathbf{n}_a\,, \tag{2.21}$$

where $\mathbf{b}_a$ is the bias, and $\mathbf{n}_a$ is the measurement noise.

3

Continuous-time visual-inertial fusion

This chapter describes methods for reconstructing the 3D world using images captured by a camera, and also aided by inertial measurements from an IMU. While this has classically been done using a discrete trajectory formulation, this thesis uses an alternative continuous-time formulation that also allows for the use of rolling shutter measurements.

The chapter starts by describing the general simultaneous localization and mapping formulation, and then describes the structure from motion problem as a special case of this. The standard bundle adjustment formulation of structure from motion is then expanded to its continuous-time formulation. Finally, it is shown how this allows for fusion with inertial measurements.

3.1 Simultaneous localization and mapping (SLAM)

In robotics one key problem is called *simultaneous localization and mapping*, or *SLAM*. The idea is that a robot inserted into an unknown environment should be able to create a map of its surroundings, while also keeping track of its own position relative to the map [14]. To do so, the robot can use measurements from whatever sensors it has access to, e.g. radar, LIDAR, camera, or IMU.

Figure 3.1 shows a graph representation of the SLAM problem. Here, $\mathbf{a}_j$ is the robot state (e.g. position) at time j. The map, $\mathbf{m}$, consists of *landmarks*, $\mathbf{m}_k$, which are observed at different times, j, to form *measurements* $\mathbf{z}_{k,j}$. The state transition $\mathbf{a}_j \to \mathbf{a}_{j+1}$ depends on the *control signal* $\mathbf{u}_j$ (e.g. "turn 30° left" or "accelerate"). The SLAM problem is now to find both the map and the robot state(s), given the measurements and control signals.

The robot state, $\mathbf{a}_j$, contains the current beliefs about the robot. This (usually) includes its position relative to the map, but could also include e.g. its velocity and orientation.

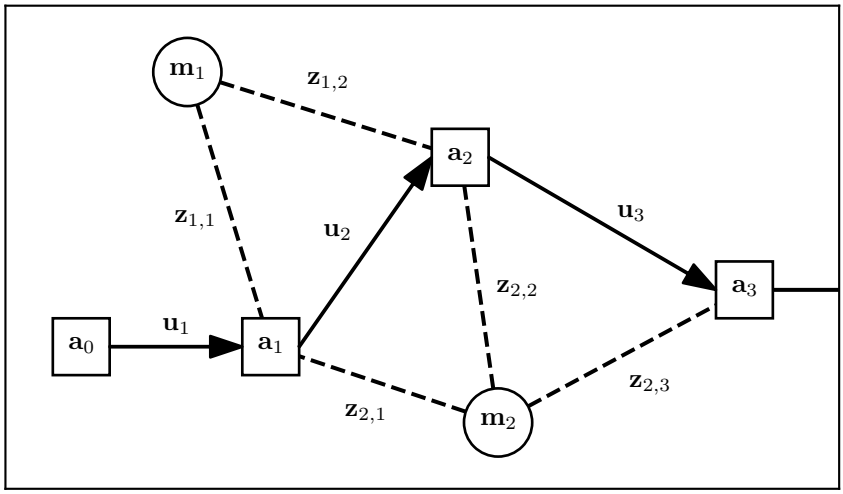


Figure 3.1: Graph representation of the SLAM problem. States $\mathbf{a}_j$ are connected to each other, and depend on the control signals, $\mathbf{u}_j$. Landmarks $\mathbf{m}_k$ are observed at different times j, generating measurements $\mathbf{z}_{k,j}$.

The map, $\mathbf{m}$, is (usually) a set of landmark locations, where a landmark is a part of the surroundings that can be robustly detected by the sensors on the robot. The important part is not what a landmark is (e.g. a tree, a rock, or a door), but that it is static in the scene, and that the sensor is able to redetect or track it as the robot is moving.

The nature of the measurements, $\mathbf{z}_{k,j}$, depends on the type of sensor that is involved. For example, a radar sensor might provide measurements $\mathbf{z}_{k,j} = (\theta, d)$: the bearing and distance to the landmark, while a camera provides the image plane location, $\mathbf{z}_{k,j} = \mathbf{y}_{k,j}$, of the imaged landmark (see section 2.1).

A common approach to solve the SLAM problem is *probabilistic SLAM*, where the goal is to find the parameters that maximize the joint posterior distribution [14]

$$p(\mathbf{a}_j, \mathbf{m} \,|\, \mathbf{Z}_{0:j}, \mathbf{U}_{0:j}, \mathbf{a}_0)\,. \tag{3.1}$$

Here, the notation $\mathbf{X}_{0:j}$ should be read as "the set of all x up to time j", and $\mathbf{a}_0$ is the initial robot state.

Filtering vs. smoothing

In a robotics context, the SLAM problem usually has a tight time budget. The robot needs to know where it is, and where it is supposed to go (and not go!), and it needs that information updated at all times. If the localization

and mapping process takes too long, the robot may have to stop and let the computations finish before it can start moving again. Depending on the type of robot and context, this is either a nuisance or not possible. For this reason it is common to only be interested in the current robot state, $\mathbf{a}_j$, in what is called the *filtering* approach. One common filtering technique is the extended Kalman filter (EKF) [24].

If the full robot trajectory is desired, one can include all previous robot states, $\mathbf{A}_{0:j-1}$, in the current state, which is known as *smoothing* [24]. A downside of smoothing is that the current state, $\mathbf{a}_j$, grows over time, which increases the problem complexity. However, it has the added benefit of being able to incorporate previous knowledge of the robot location in the update, by reestimating previous states given new information, which can improve accuracy.

3.2 Structure from motion by bundle adjustment

A related problem in computer vision is the *structure from motion* (SfM) problem, which has its roots in photogrammetry. Given a set of N (unordered) images $\mathcal{I} = \{I_n : n = 1 \dots N\}$, the goal is to find both the 3D structure of the scene, and also the camera poses

$$\mathbf{T}_n = (\mathbf{R}_n, \mathbf{p}_n), \tag{3.2}$$

where $\mathbf{R}_n \in \mathbb{SO}(3)$ is the camera orientation, and $\mathbf{p} \in \mathbb{R}^3$ is its location. This is similar to the more general SLAM problem, but more emphasis is placed on mapping (structure) than on localization[1].

The camera pose $\mathbf{T}_n$ is defined as the transformation

$$\begin{aligned} \mathbf{x}^w &= \mathbf{R}_n \mathbf{x}^c + \mathbf{p}_n \\ &\sim \begin{bmatrix} \mathbf{R}_n & \mathbf{p}_n \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}^c \\ 1 \end{bmatrix}, \end{aligned} \tag{3.3}$$

which moves a 3D point $\mathbf{x}^c$ in the camera coordinate frame, to the point $\mathbf{x}^w$ in the world coordinate frame. Since the coordinate frames are usually obvious from the context, the indices w and c are often omitted for brevity. With a slight abuse of notation, the transformation will sometimes be written as $\mathbf{T}_n\mathbf{x}$, where the homogeneous representation of $\mathbf{x}$ is implied.

The structure from motion goal is then to estimate both the set of image poses $\mathcal{T} = \{\mathbf{T}_n : n = 1 \dots N\}$, and the set of 3D points $\mathcal{X} = \{\mathbf{x}_k : k = 1 \dots K\}$ given a set of image measurements $\mathcal{Y} = \{\mathbf{y}_{k,n} \in \mathcal{Y}_1 \cup \dots \cup \mathcal{Y}_K\}$. The tracks $(\mathcal{Y}_1, \dots, \mathcal{Y}_K)$ are typically computed using the feature matching approach outlined in section 2.1.

[1] SfM is sometimes referred to as *structure and motion* (SaM), which emphasizes that the goal is to also retrieve the trajectory/motion.

The preferred solution to the structure from motion problem is *bundle adjustment* [69]. Here, the objective is to minimize the cost function

$$J(\mathcal{T}, \mathcal{X}) = \sum_{\mathbf{T}_n \in \mathcal{T}} \sum_{\mathbf{x}_k \in \mathcal{X}} \|\mathbf{y}_{k,n} - \pi(\mathbf{T}_n^{-1}\mathbf{x}_k)\|^2 , \tag{3.4}$$

where $\pi(\cdot)$ is the projection function defined in (2.12). By assuming independence, and Gaussian probability distributions, for the measurements, it can be shown [69] that (3.4) is the maximum likelihood solution to the SLAM problem in (3.1).

One difference between SLAM and SfM is that the former includes control signals. Since bundle adjustment generally assumes that the images are unordered, the concept of control signals makes little sense here. However, this thesis uses video as the source for image data, and thus the images are in fact ordered. Therefore something equivalent to the control signals can be used, which will be discussed later.

Solving the bundle adjustment problem

To minimize $J(\mathcal{T}, \mathcal{X})$ in (3.4), one typically uses a non-linear least-squares method, such as Levenberg-Marquardt [43]. The bundle adjustment problem in (3.4) is generally non-convex, which means that the found solution depends on an initial estimate of the camera poses and 3D structure. If the initial estimate is too far from the optimum, there is a large risk that a local minimum will be found, instead of the global one.

One way to seed the system with a good initial estimate is to perform *incremental* SfM. Instead of solving for all 3D points and camera poses at once, the problem is extended incrementally by adding new images and 3D points successively. Using an initial image pair, the relative pose between the corresponding cameras can be found by estimating the *essential matrix* (see section 3.8). An initial set of 3D points can then be triangulated from this solution. When a new image is added to the problem, its pose $\mathbf{T}_n$ is estimated by solving the *perspective-N-point problem*, and new points visible in more than one image are triangulated. Finally, the bundle adjustment problem is solved for the new, larger, set of images and 3D points.

In contrast, *global* SfM instead tries to estimate all 3D points and camera poses at once. One approach [45, 16] is to first use epipolar geometry to find pairwise relative poses between a large number of images. These are then used to compute the absolute orientations and translations of the camera poses.

Using the incremental approach it is possible to use bundle adjustment also for online applications, and in this case it is similar to the smoothing approach to SLAM. Every added image makes the bundle adjustment problem grow larger, increasing the time it takes to solve it. Instead of estimating all image poses, a smaller subset of the images, called *keyframes*, can be selected to represent the larger problem, and keep computational complexity

down. Keyframing only reduces the problem of infinite state growth, which motivates the addition of a *sliding window* approach where only the m most recent images (or keyframes) are included in the bundle adjustment problem.

3.3 Continuous-time structure from motion

In the bundle adjustment problem we assigned one camera pose, $\mathbf{T}_n$, to each camera view, which implies that all measurements in this image share this single pose. This is true for a camera with a global shutter, but not when the camera has a rolling shutter, as described in section 2.1. The naive solution is to add one camera pose *per image row*, but this makes the problem intractable since the number of camera poses grows too large. Also, few of these row-poses would be associated with more than one measurement, which would make the problem ill-posed. Another approach is to ignore the rolling shutter, and (erroneously) assume a global shutter model, but this has been shown to lead to reconstruction failures [27].

Instead of modeling the camera trajectory as a set of discrete poses, it can be represented as a continuous function

$$\mathbf{T}(t) = (\mathbf{R}(t), \mathbf{p}(t)) \,. \tag{3.5}$$

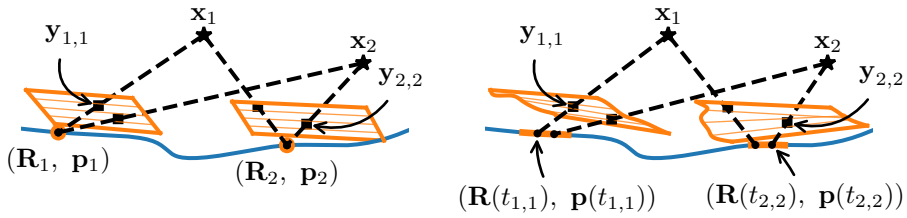
While trajectory usually means position, it is here used to mean a sequence of poses, which are defined as in (3.3). Since the continuous formulation can provide a pose at any time instance t, it is a convenient way to handle the problem of rolling shutter measurements. Using a continuous-time trajectory, we modify the bundle adjustment cost function (3.4) to

$$J(\mathcal{T}, \mathcal{X}) = \sum_{\mathbf{y}_{k,n} \in \mathcal{Y}} \|\mathbf{y}_{k,n} - \pi(\mathbf{T}^{-1}(t_{k,n})\mathbf{x}_k)\|^2 \,, \tag{3.6}$$

where $\mathcal{Y}$ is the set of all measurements, and $t_{k,n}$ is the time which corresponds to the measurement $\mathbf{y}_{k,n}$. In the discrete case, $\mathcal{T}$ was the set of all camera poses, but here it is changed to the parameters of the continuous-time trajectory, $\mathbf{T}(t)$. The exact nature of these parameters depends on the chosen continuous-time trajectory model, which will be explained shortly.

Figure 3.2a shows the discrete case of structure from motion, where each camera has a single associated camera pose. Since the cameras have global shutters, all image rows share a single camera pose, and the image plane is rectangular, as expected. In contrast, figure 3.2b shows the continuous case, with a rolling shutter camera. Here, each row has its own camera pose, indicated by the overlay on the trajectory, which results in a warped image plane.

The continuous formulation can be used not only to handle rolling shutter camera measurements, but is a convenient choice also for other types of sensors, e.g. radar [7], LIDAR [5, 8], and event cameras [46]. Later, in section



(a) Discrete SfM (global shutter) (b) Continuous-time SfM (rolling shutter)

Figure 3.2: The bundle adjustment problem visualized for the discrete case in (3.4), and the continuous-time case in (3.6). The 3D points $\mathbf{x}_k$ are projected as image measurements $\mathbf{y}_{k,n}$ in the two images $n \in \{1, 2\}$. In the continuous-time case the camera uses a rolling shutter, which results in warped image planes. The solid line represents the true camera motion, and overlaid on this is the time during which the image was integrated, as well as the time instance that belongs to each measurement (black dot)

3.4, it is shown how the continuous formulation can be used to fuse visual and inertial measurements in a natural way.

Continuous-time trajectory models

The continuous trajectory $\mathbf{T}(t)$ can be modeled in many different ways. One common approach is to model the trajectory as a combination of temporal basis functions, or similarly, as *splines* [20, 40, 8, 4, 5, 33, 49, 46]. The spline approach is used in Paper C and Paper D, and is covered in chapter 4.

Hedborg et al. [27] addressed the rolling shutter problem in bundle adjustment by linear interpolation between adjacent camera poses, $\mathbf{T}_n$ and $\mathbf{T}_{n+1}$, such that

$$\mathbf{R}(t) = \texttt{SLERP}(s(t), \mathbf{R}_n, \mathbf{R}_{n+1}) \,, \tag{3.7}$$

and

$$\mathbf{p}(t) = \mathbf{p}_n + s(t)(\mathbf{p}_{n+1} - \mathbf{p}_n) \,. \tag{3.8}$$

Here, $\texttt{SLERP}(\cdot)$ is spherical linear interpolation (see [65], and (4.28)), and

$$s(t) = \frac{t - t_n}{t_{n+1} - t_n} \in [0, 1] \tag{3.9}$$

is the interpolation parameter. Although no explicit trajectory estimation is performed there, the interpolation approach is also used in Paper A and Paper B.

A third approach is to model continuous trajectories as *Gaussian processes* [3, 68].

3.4 Inertial measurements

While standard structure from motion only concerns itself with visual measurements, inertial measurements can be a valuable addition for certain problems. The question is then how to incorporate the inertial measurements into the structure from motion problem.

In SLAM, one possibility is to use the inertial measurements as control signals (i.e. $\mathbf{U}_{0:k}$ in (3.1)). This way the inertial measurements drive the update of the robot state. Instead of updating the state on every inertial measurement, another approach is to instead integrate the measurements between a sparser set of poses, which are further apart in time. Paper B uses integration of a gyroscope signal to find the camera orientation, and then uses visual measurements to find the calibration parameters of the camera-IMU system which best explains this orientation sequence (see chapter 5). A problem with this type of integration is that the inertial measurements are corrupted by a measurement bias (see section 2.3) which means that the integration must be recomputed every time the estimate of the bias (or any other dependent parameter) changes. A way to reduce the need for re-integration is to use the preintegration theory developed by Forster et al. [19].

Instead of using the inertial measurements as control signals to steer the state updates, the measurements can be included in the bundle adjustment cost function, just like the visual measurements. This is the approach used by Furgale et al. [20], Lovegrove et al. [40], and Paper C and Paper D. Here, the cost function in (3.6) is augmented by adding error terms also for the accelerometer and gyroscope:

$$\begin{aligned} J(\mathcal{T},\mathcal{X}) = \sum_{\mathbf{y}_{k,n}\in\mathcal{Y}} \underbrace{\|\mathbf{y}_{k,n} - \pi(\mathbf{T}^{-1}(t_{k,n})\mathbf{x}_k)\|^2_{\mathbf{W}_v}}_{\text{Visual}} \\ + \sum_m \underbrace{\|\boldsymbol{\omega}_m - \nabla_\omega \mathbf{T}(t_m)\|^2_{\mathbf{W}_g}}_{\text{Gyroscope}} \\ + \sum_l \underbrace{\|\mathbf{a}_l - \nabla_a^2 \mathbf{T}(t_l)\|^2_{\mathbf{W}_a}}_{\text{Accelerometer}} . \end{aligned} \tag{3.10}$$

The operators ∇_ω and ∇_a^2 are used to predict gyroscope and accelerometer measurements, respectively, from the continuous-time trajectory, by using analytical derivation. Exact expressions of ∇_ω and ∇_a^2 depend on the choice of both the trajectory model, and IMU measurement models, where two specific examples can be found in Paper D.

The inertial measurements are typically not sampled synchronously with the image measurements (they usually have a much higher sample rate), and their predictions require derivatives of $\mathbf{T}(t)$. Because of this, a continuous-time trajectory is not only nice to have, but a requirement, in this case. To

support prediction of accelerometer measurements, the trajectory function $\mathbf{T}(t)$ must be at least twice differentiable.

Balancing the measurements

The error terms, or residuals, in the cost function (3.10) are associated with their respective *weight matrices*, $\mathbf{W}_v$, $\mathbf{W}_g$ and $\mathbf{W}_a$, such that

$$\|\mathbf{r}\|^2_{\mathbf{W}} = \mathbf{r}^T\mathbf{W}\mathbf{r}\,. \tag{3.11}$$

In the previous cost functions, (3.4) and (3.6), where only visual measurements were available, the weight is assumed to be $\mathbf{W}_v = \mathbf{I}$, which results in a standard non-linear least-squares problem. When different residuals instead use different weights, the problem is transformed to a weighted non-linear least-squares problem (WNLS). The weights are required to achieve a fair balancing between the measurements. In (3.10), the residuals have different units of measurement, i.e. pixels, rad/s, and m/s^2, as well as different noise characteristics. Therefore, to assume equal residual weights here would put almost all weight on the visual measurements, which is not desirable.

The standard approach in WNLS, which is used by Lovegrove et al. [40], is to set $\mathbf{W} = \mathbf{\Sigma}^{-1}$, where $\mathbf{\Sigma}$ is the covariance matrix of the measurement noise. The covariance can either be taken from the data sheet of the sensor, or estimated offline in a calibration procedure. However, this weighting makes the implicit assumption that the trajectory function $\mathbf{T}(t)$ is able to perfectly model the motion of the sensor. In general applications it is not feasible to have a perfect representation of the trajectory, which means that all the measurement residuals will be the combination of two error terms: the usual measurement noise, and the *trajectory approximation* noise. The latter depends on the parameterization of the trajectory. Paper C presents a way to estimate the trajectory approximation error for a splined trajectory, and use this to better weight the inertial measurements. The trajectory approximation problem is covered in more detail in chapter 4.

3.5 Trajectory smoothness constraints

One problem when estimating a trajectory is how to avoid physically meaningless solutions. For example, a human on foot is unlikely to travel faster than 10 m/s, and would perish under excessive accelerations. The rolling shutter effectively increases the sample rate of the visual measurements by several orders of magnitude. If the trajectory model is not constrained it is usually possible to find a preposterous solution due to overfitting. To avoid overfitting, it is important that a continuous-time trajectory estimator puts constraints on the allowed motions. In some cases, in the form of a *motion prior*, which is a probability distribution that describes the expected kinematics. Degeneracies in bundle adjustment due to rolling shutter modelling

have previously been studied by Albl et al. [2], who showed the importance of using a constrained camera motion.

The most common smoothness constraint found in the continuous-time trajectory estimation literature, is to put limits on the allowed acceleration. Lovegrove et al. [40], who use a splined trajectory, includes the linear and angular accelerations at each spline knot as extra terms in the least-squares cost function. Others [20, 4, 49] use a kinematic model with acceleration

$$\ddot{\mathbf{p}}(t) = \mathbf{n}(t), \tag{3.12}$$

where $\mathbf{n}(t)$ is a zero-mean Gaussian process.

If inertial measurements are available, then these can also be used to constrain the allowed trajectory. As discussed in section 3.4, these can either be used as control signals to update the robot state, or be included in the bundle adjustment cost function. Both options constrain the the trajectory by ensuring that it does not disagree too much with the inertial measurements.

3.6 Landmark representation

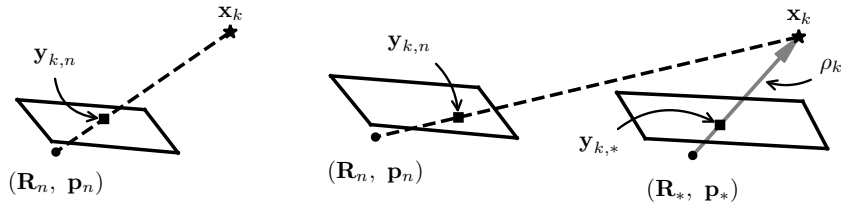
In the standard bundle adjustment formulation, the landmarks in $\mathcal{X}$ are represented as Cartesian 3D points $\mathbf{x}_k \in \mathbb{R}^3$. These are projected into images using the associated camera pose (whether continuous or discrete), as illustrated in figure 3.3a.

Another option is the inverse depth formulation used in Lovegrove et al. [40], Paper C and Paper D. Here the landmark k is instead represented by $(\mathbf{y}_{k,*}, \rho_k)$, where $\mathbf{y}_{k,*}$ is the observation of the landmark in a *reference image*, I_*, and ρ_k is its inverse depth. Figure 3.3b shows how an (implicit) 3D point, $\mathbf{x}_k$, is created using the measurement and pose at the reference image, together with the inverse depth. The newly formed $\mathbf{x}_k$ is then projected into camera n, to form the measurement, $\mathbf{y}_{k,n}$. This process can be expressed by the *transfer function* $\psi(\cdot)$, which is defined as

$$\begin{aligned} \mathbf{y}_{k,n} &= \psi(\mathbf{y}_{k,*}, \mathbf{T}^{-1}(t_{k,n})\mathbf{T}(t_{k,*}), \rho_k) \\ &= \pi\left(\mathbf{T}^{-1}(t_{k,n}) \underbrace{\mathbf{T}(t_{k,*}) \begin{bmatrix} \pi^{-1}(\mathbf{y}_{k,*}) \\ \rho_k \end{bmatrix}}_{\mathbf{x}_k} \right). \end{aligned} \tag{3.13}$$

Since the reference observation $\mathbf{y}_{k,*}$ is fixed, only the scalar ρ_k needs to be estimated. This reduces the size of each landmark from three to one parameters, which thus reduces the size of the bundle adjustment problem. However, errors in $\mathbf{y}_{k,*}$ now risk introducing biases in the other measurements, and there is also one less measurement to use.

By setting $\rho_k = 0$, the inverse depth formulation can place points at infinity. While these points are not useful in the final 3D structure, they still



(a) Cartesian (b) Inverse depth

Figure 3.3: The projection of a landmark $\mathbf{x}_k$ into an image point $\mathbf{y}_{k,n}$ in camera n, using two different landmark representations. In (a) the landmark is represented by a Cartesian 3D point. In (b) the landmark is represented by ρ_k, the inverse depth of the landmark, with respect to the reference image, $*$.

give useful information about the relative orientation of cameras which are observing it. Allowing points at infinity can be used to avoid having to use triangulation to initialize new landmarks. New landmarks can instead first be placed at infinity, where they can provide useful information even before they are upgraded to real, finite, points.

A third approach is to model a landmark as the homogeneous vector $\mathbf{x}_k = \begin{bmatrix} x & y & z & w \end{bmatrix}$ [69]. Just like the inverse depth formulation, this can handle points at infinity, i.e. by setting $w = 0$.

3.7 Rolling shutter modelling

One of the main reasons for using a continuous-time trajectory is to handle rolling shutter measurements. As described in section 2.1, the rolling shutter causes the image to be captured row-by-row, which means that every row has its own camera pose. The observation of landmark k in image n is defined as

$$\mathbf{y}_{k,n} = \begin{bmatrix} u \\ v \end{bmatrix} = \pi(\mathbf{T}^{-1}(t_{k,n})\mathbf{x}_k)\,, \tag{3.14}$$

where the projection time is

$$t_{k,n} = t_n + r\frac{v}{N_v}\,. \tag{3.15}$$

Here, r is the rolling shutter image readout time, t_n is the time of the first row of image n, and N_v is the number of image rows.

The problem with (3.14) and (3.15) is that they form a cyclic dependency:

1. To compute the projection time, $t_{k,n}$, using (3.15) we need the row, v, from (3.14), but

2. to compute the row, v, using (3.14), we need the projection time, $t_{k,n}$, from (3.15).

This chicken and egg problem means that there is no general closed form solution for rolling shutter projection[2]. Different options for solving the rolling shutter projection have been proposed, and one of the contributions of Paper D is to compare three of these methods.

The rolling shutter projection methods can be characterized by how they handle the *rolling shutter time deviation*

$$\epsilon(t_{k,n}) = (t_{k,n} - t_n)\frac{N_v}{r} - \pi_v(t_{k,n})\,, \tag{3.16}$$

where $\pi_v(t_{k,n})$ is short hand notation for the row of the projection in (3.14). The rolling shutter time deviation can similarly be defined using the transfer function from section 3.6, by changing $\pi_v(t_{k,n})$ to $\psi_v(t_{k,n})$.

The ideal case (i.e. with no measurement noise) is a method that satisfies the *rolling shutter constraint* [23]

$$\epsilon(t_{k,n}) = 0\,. \tag{3.17}$$

If this constraint does not hold, it means that the rolling shutter model is no longer valid. The only case when the rolling shutter constraint must hold, is when synthetic measurements are generated as part of an experiment. If not, the resulting data is not a good representation of a rolling shutter camera. In this thesis, the preferred way to generate synthetic rolling shutter measurements is to solve (3.17) using a root finding algorithm, e.g. the one by Brent [9].

Paper D shows that while there is a trade off between accuracy (i.e. satisfying the constraint) and time complexity, the slower and more exact methods do not give any advantage in the context of structure from motion.

3.8 Robust structure from motion

To solve the bundle adjustment problems in (3.4), (3.6), and (3.10), the main assumption is that the errors in the image measurements are normally distributed, with zero mean. However, visual measurements created by tracking or feature matching are subject to association errors, where the image measurements in a track $\mathcal{Y}_k$ do not belong to the same object. These *outlier* measurements need to be removed, or the bundle adjustment solution will be biased.

[2] Solutions for certain types of motions have been suggested by Geyer et al. [23].

Geometric consistency

In standard bundle adjustment with global shutter measurements (i.e. (3.4)), the main method for removing the outliers is to enforce *geometric consistency*. Given two images, a and b, valid measurements should satisfy the *epipolar constraint*

$$\pi^{-1}(\mathbf{y}_{k,a})^T \mathbf{E} \pi^{-1}(\mathbf{y}_{k,b}) = 0\,, \tag{3.18}$$

where $\mathbf{E}$ is the essential matrix, and $\pi^{-1}(\mathbf{y})$ are normalized image coordinates [25]. The essential matrix is defined by the relative pose between the two images. Given image poses $\mathbf{T}_a = (\mathbf{I}, \mathbf{0})$ and $\mathbf{T}_b = (\mathbf{R}, \mathbf{p})$, the essential matrix can be computed as

$$\mathbf{E} = [\mathbf{p}]_\times \mathbf{R}\,, \tag{3.19}$$

where $[\cdot]_\times$ is the cross-product operator, such that $[\mathbf{a}]_\times \mathbf{b} = \mathbf{a} \times \mathbf{b}$. By estimating the essential matrix, and thus the relative pose, e.g. by using random sample consensus (RANSAC) [17], the measurements that do not satisfy the epipolar constraint can be removed.

Since the essential matrix assumes a single pose per image, it can not be directly applied to rolling shutter measurements. Rolling shutter versions of the relative pose problem have however been proposed, e.g. "R6P" by Albl et al. [1], and the work by Dai et al. [10]. Both these methods make assumptions on the motion of the camera during the time which the images were captured, where constant angular and linear velocity is the most permissive case. While these assumptions may be useful for many applications, they do not hold in general. These methods also suffer from relatively high complexity, by either returning many solutions, or requiring up to 44 corresponding image points. This requires a substantially higher number of RANSAC iterations, compared to the global shutter case.

Since enforcing geometric consistency is not feasible in the rolling shutter case, the cross-checking by backwards tracking, as described in section 2.1, becomes more important.

Robust loss functions

When outlier rejection by geometric consistency is impossible, or at least difficult, a second option is to instead accept the presence of outliers and try to mitigate their effect. In the standard bundle adjustment problem, each image measurement residual $\mathbf{r} = \begin{bmatrix} r_u & r_v \end{bmatrix}^T$ adds the two terms $r_u^2 + r_v^2$ to the cost function. These terms can instead be rewritten as $\phi(r_u) + \phi(r_v)$ where $\phi(r)$ is a *loss function* [73]. The standard least squares loss function

$$\phi_{\text{squared}}(r) = r^2\,, \tag{3.20}$$

will put a very large weight on outliers, since these usually have large reprojection errors. To avoid this we can instead use a robust loss function, such as the one by Huber [29], which is defined as

$$\phi_{\text{Huber}}(r, a) = \begin{cases} r^2 & r \leq a\,, \\ 2a(|r| - \frac{1}{2}a) & r > a\,. \end{cases} \tag{3.21}$$

The interpretation of the Huber loss function on image measurements is that residuals with a reprojection error smaller than a pixels are treated exactly like with the standard, squaring, loss function. Measurements with residuals larger than a are considered to be outliers, and are instead weighted linearly, which makes their impact less critical.

There exists also loss functions which act on the whole residual, to form a combined loss, see e.g. Engels et al. [15].

4

Spline-based trajectories

While a continuous-time trajectory can be represented in many ways, one convenient approach is to model it as a spline. A splined trajectory is simple to reason about, and also to differentiate analytically. Differentiation is important since it allows predictions of inertial measurements.

In his chapter, splines are first introduced in their simplest scalar form. Later it is shown how splines can be constructed also for $\mathbb{R}^3$, $\mathbb{SO}(3)$, and $\mathbb{SE}(3)$, and how these can be used to represent a camera pose trajectory. Finally, this chapter covers the problem of how to select the spline density, and how to model the trajectory approximation error mentioned in section 3.4.

4.1 B-splines

A spline is a function defined by piecewise polynomials, where each of the segments are connected in such a way that the function is smooth. The connection points, t_k, between the segments are known as *knots*, and the spline *degree* is the maximum degree of all segment polynomials.

Instead of directly defining the polynomials of the spline segments, the *basis spline*, or *B-spline*, is defined as the weighted sum

$$f(t) = \sum_{k=1}^{K} B_{k,n}(t) c_k \,. \tag{4.1}$$

The *control points*, c_k, are weighted by their respective *basis functions*, $B_{k,n}(t)$, where the *order*, n, indicates that the spline (and thus the basis functions) have degree $n-1$. Using the *knot vector* $\begin{bmatrix} t_1 & t_2 & \dots & t_K \end{bmatrix}$, and the Cox-de Boor [11] recursion formula, the B-spline basis functions can be

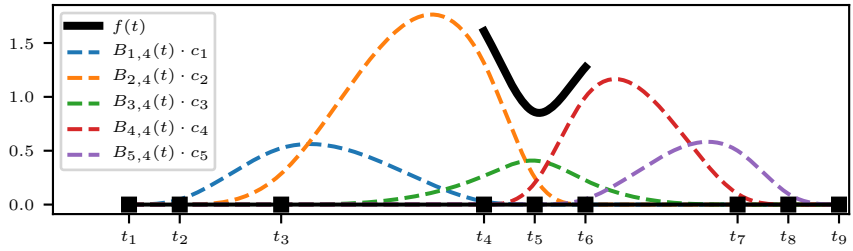


Figure 4.1: An example of a cubic B-spline, $f(t)$, with a non-uniform knot vector (black squares). The dashed lines are the product of the basis functions with their respective control point. Only the five basis functions which are non-zero on the two valid spline segments are shown. The control points for these basis functions are set to 1, 3, 0.5, 2, and 1, respectively.

defined as

$$B_{k,n}(t) = \frac{t - t_k}{t_{k+n-1} - t_k} B_{k,n-1}(t) + \frac{t_{k+n} - t}{t_{k+n} - t_{k+1}} B_{k+1,n-1}(t)\,, \qquad \text{where} \tag{4.2}$$

$$B_{k,1}(t) = \begin{cases} 1 & \text{if } t \in [t_k, t_{k+1}) \\ 0 & \text{otherwise} \end{cases}. \tag{4.3}$$

Figure 4.1 shows an example of a cubic ($n = 4$) B-spline. In this figure, only the two middle segments are valid, since a segment $t \in [t_k, t_{k+1})$ depends on knots $(t_{k-n+1}, \ldots, t_{k+n})$. This means that a segment must be both preceded and succeeded by at least $n - 1$ knots.

The B-spline basis functions have local support, which means that only the n basis functions $(B_{k-n+1,n}(t), \ldots, B_{k,n}(t))$ are non-zero when evaluating the spline in the segment $t \in [t_k, t_{k+1})$. This property is very important in the context of trajectory representation, since it means that a measurement at time t depends on only n control points, and not all of them. If this was not the case, then solving the structure from motion problem in chapter 3 would be intractable.

Uniform B-splines

If the spline knots are not placed arbitrarily, but are instead evenly spaced, such that

$$t_k = k\Delta t\,, \tag{4.4}$$

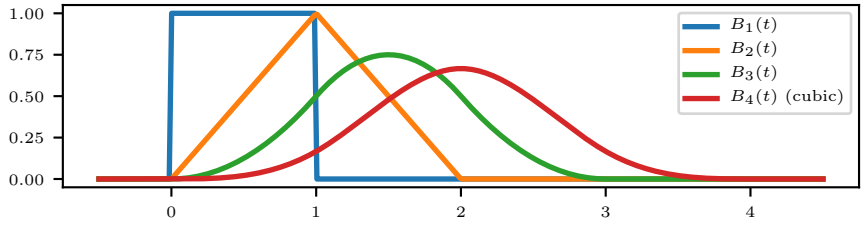


Figure 4.2: Uniform B-spline basis functions, $B_n(t)$, for different spline orders, n, with spline knot spacing $\Delta t = 1$. The support of each basis function is exactly n knots.

then the spline is said to be *uniform* with *knot spacing* $\Delta t = t_{k+1} - t_k$ for all k. The uniform B-spline simplifies the expression in (4.1) to

$$f(t) = \sum_{k=1}^{K} B_n(t - k\Delta t)c_k \,. \tag{4.5}$$

The basis functions are now shifted copies of a single basis function, $B_n(t)$, which is defined only by the spline order. Figure 4.2 shows the basis functions for different orders of uniform B-splines. This figure also illustrates the local support property.

Uniform B-splines are convenient since they are defined fully by their order, n, the control points, $(c_1, \ldots, c_K)$, and the knot spacing, Δt.

Matrix form

A convenient way to implement splines is to use the matrix formulation by Qin [61]. To find the spline value for $t \in [t_k, t_{k+1})$, we first define the interpolation parameter

$$u(t) = \frac{t - t_k}{t_{k+1} - t_k} \in [0, 1] \,. \tag{4.6}$$

The values of the n basis functions which have support for the selected t are then expressed as the vector

$$\begin{aligned} \mathbf{B}_{k,n}(u) &= \begin{bmatrix} B_{k-n+1,n}(u) & \ldots & B_{k-1,n}(u) & B_{k,n}(u) \end{bmatrix} \\ &= \mathbf{u}^T \mathbf{M}^n(k) = \begin{bmatrix} 1 & u & u^2 & \ldots & u^{n-2} & u^{n-1} \end{bmatrix} \mathbf{M}^n(k) \,. \end{aligned} \tag{4.7}$$

The matrix $\mathbf{M}^n(k) \in \mathbb{R}^{n \times n}$ depends only on the knot placements, and the current spline segment, $[t_k, t_{k+1})$. The $\mathbf{M}^n(k)$ matrices can therefore be precomputed and kept constant, until the knot vector changes. The value of the spline can now be computed by

$$f(t) = \mathbf{B}_{k,n}(u)\mathbf{c} = \mathbf{B}_{k,n}(u) \begin{bmatrix} c_{k-n+1} & \ldots & c_{k-1} & c_k \end{bmatrix}^T \,. \tag{4.8}$$

Derivatives of the splines are easily computed by differentiating the $\mathbf{u}$ vector:

$$\frac{d^n}{dt^n} f(t) = \left(\frac{d^n}{dt^n} \mathbf{u}(u(t))^T \right) \mathbf{M}^n(k)\mathbf{c}\,. \tag{4.9}$$

To predict IMU measurements, only the first two derivatives are of interest, and these are given here as

$$\frac{d}{dt} f(t) = \frac{1}{\Delta t_k} \begin{bmatrix} 0 & 1 & 2u & \dots & (n-1)u^{n-2} \end{bmatrix} \mathbf{M}^n(k)\mathbf{c}\,, \quad \text{and} \tag{4.10}$$

$$\frac{d^2}{dt^2} f(t) = \frac{1}{\Delta t_k^2} \begin{bmatrix} 0 & 0 & 2 & \dots & (n-1)(n-2)u^{n-3} \end{bmatrix} \mathbf{M}^n(k)\mathbf{c}\,, \tag{4.11}$$

where $\Delta t_k = t_{k+1} - t_k$.

In the case of uniform B-splines, the matrix $\mathbf{M}^n(k) = \mathbf{M}^n$ is constant, since the knot placement is fully defined by the constant knot spacing, Δt. For a uniform, cubic ($n = 4$), B-spline, the spline matrix is defined as

$$\mathbf{M}^4 = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}. \tag{4.12}$$

4.2 Splines as camera pose trajectories

The splines described in section 4.1 were defined as scalar, 1D-functions, $f(t) \in \mathbb{R}$. To be useful as a representation for a continuous-time camera pose trajectory, we need to change the interpolation space to something more useful. The two main choices for camera pose trajectory representation are to either define a spline on the group $\mathbb{SE}(3)$, or use two separate splines on $\mathbb{R}^3$ and $\mathbb{SO}(3)$. How to construct these splines are described in sections 4.3, 4.4, and 4.5.

When a spline is used as a trajectory in continuous-time structure from motion (i.e. (3.6) or (3.10)), the trajectory parameters are usually the control points

$$\mathcal{T} = \{\mathbf{c}_1, \dots, \mathbf{c}_K\}\,. \tag{4.13}$$

The vector $\mathbf{c}_k \in \mathbb{R}^p$ is a representation of an element in the interpolation space ($\mathbb{R}^3$, $\mathbb{SO}(3)$, or $\mathbb{SE}(3)$), which is suitable for the non-linear least-squares solver. In principle, one could include also the knot vector in $\mathcal{T}$, but this is not done in the structure from motion context. Instead, the knot vector is decided on before solving the problem. This is the topic of section 4.6.

4.3 Splines in $\mathbb{R}^3$

The continuous-time trajectory function $\mathbf{T}(t)$ can be defined by using two separate splines: $\mathbf{p}(t) \in \mathbb{R}^3$ and $\mathbf{R}(t) \in \mathbb{SO}(3)$. This choice is named *split* interpolation (or representation) in Paper D.

Defining a spline in $\mathbb{R}^3$ is trivial. Since $\mathbb{R}^3$ is a linear space, one can simply use (4.1), where the control points are changed to $\mathbf{p}_k \in \mathbb{R}^3$. This results in the positional trajectory

$$\mathbf{p}(t) = \sum_{k=1}^{K} B_{k,n}(t)\mathbf{p}_k \,. \tag{4.14}$$

4.4 Splines in $\mathbb{SO}(3)$

Rotation matrices are a subset of the 3×3 matrices with elements

$$\{\mathbf{R} \in \mathbb{R}^{3\times 3} \;:\; \mathbf{R}^T\mathbf{R} = \mathbf{I}, \det \mathbf{R} = +1\} \,. \tag{4.15}$$

Under the operation of matrix multiplication, the rotation matrices form the *special orthogonal group*, $\mathbb{SO}(3)$. This group is not linear, and for some choice of $\mathbf{R}_1, \mathbf{R}_2 \in \mathbb{SO}(3)$ we have, in general,

$$(\mathbf{R}_1 + \mathbf{R}_2) \notin \mathbb{SO}(3) \,. \tag{4.16}$$

Because of this, the weighted sum expression in (4.1) can not be used. Instead, a solution based on the exponential map is outlined here.

Exponential maps on $\mathbb{SO}(3)$

$\mathbb{SO}(3)$ is also a *matrix Lie group*, and has an associated *Lie algebra*, $\mathfrak{so}(3)$ [6] with elements

$$\{[\mathbf{w}]_\times \in \mathbb{R}^{3\times 3} : [\mathbf{w}]_\times^T = -[\mathbf{w}]_\times, \mathbf{w} \in \mathbb{R}^3\} \,. \tag{4.17}$$

These elements are the set of all 3×3 skew-symmetric matrices

$$[\mathbf{w}]_\times = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}_\times = \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix} . \tag{4.18}$$

Strictly speaking, $\mathfrak{so}(3)$ is the *vector space* of the Lie algebra, but here it will be used to refer to the Lie algebra itself.

An element in $\mathfrak{so}(3)$ can be uniquely mapped to an element in $\mathbb{SO}(3)$ using the *exponential map*. Given the axis angle vector $\mathbf{w} = \alpha\mathbf{n}$, where $\|\mathbf{n}\| = 1$ we have

$$\begin{aligned} \mathbf{R} &= \exp([\mathbf{w}]_\times) = \exp(\alpha[\mathbf{n}]_\times) \\ &= \mathbf{I} + \sin\alpha[\mathbf{n}]_\times + (1 - \cos\alpha)[\mathbf{n}]_\times^2 \,, \end{aligned} \tag{4.19}$$

where the last expression is the well-known *Rodrigues' rotation formula* [47].

Inversely, an element in $\mathfrak{so}(3)$ can be retrieved from an element in $\mathbb{SO}(3)$ by using the logarithm

$$[\mathbf{w}]_\times = \log(\mathbf{R})\,. \tag{4.20}$$

The property of the Lie algebra that is most interesting, in the context of splines, is that all vectors $\mathbf{w} \in \mathbb{R}^3$ are associated with an element in $\mathbb{SO}(3)$. Computations can thus be performed in the Lie algebra, instead of on the group, and the result can then always be mapped back into $\mathbb{SO}(3)$.

Quaternion B-spline

A quaternion

$$\begin{aligned}\mathbf{q} &= a + xi + yj + zk \\ &= (a, \mathbf{v})\,, \quad \mathbf{v} = \begin{bmatrix} x & y & z \end{bmatrix}^T\,,\end{aligned} \tag{4.21}$$

is an extension of the complex numbers, with imaginary units, i, j, and k.

It is well known that the unit quaternion

$$\mathbf{q} = (\cos\frac{\theta}{2}, \mathbf{n}\sin\frac{\theta}{2})\,, \quad \text{where} \quad \|\mathbf{q}\| = \|\mathbf{n}\| = 1\,, \tag{4.22}$$

represents a rotation of θ radians around the rotation axis $\mathbf{n}$ [65]. Quaternions are therefore a more compact representation of a rotation, compared to a full rotation matrix with four instead of nine elements.

The equivalent of the exponential map for $\mathbb{SO}(3)$ exists also for the unit quaternions [35], with

$$\mathbf{w} = \log(\mathbf{q}) = \left(\frac{\mathbf{v}}{\|\mathbf{v}\|}\cos^{-1} a\right) \in \mathbb{R}^3\,, \tag{4.23}$$

and

$$\mathbf{q} = \exp(\mathbf{w}) = \left(\cos\|\mathbf{w}\|, \frac{\mathbf{w}}{\|\mathbf{w}\|}\sin\|\mathbf{w}\|\right)\,, \tag{4.24}$$

where $[\mathbf{w}]_\times$ is again an element of $\mathfrak{so}(3)$.

Kim et al. [36] construct quaternion splines by first noting that the $\mathbb{R}^3$ spline in (4.14) can be written on its equivalent *cumulative* form

$$\mathbf{p}(t) = \mathbf{p}_1\tilde{B}_{1,n}(t) + \sum_{k=2}^{K}(\mathbf{p}_k - \mathbf{p}_{k-1})\tilde{B}_{k,n}(t)\,, \tag{4.25}$$

where

$$\tilde{B}_{k,n}(t) = \sum_{i=k}^{K} B_{i,n}(t) \tag{4.26}$$

are *cumulative basis functions*.

By changing the sum to quaternion multiplications, a quaternion spline

$$\mathbf{q}(t) = \mathbf{q}_1^{\tilde{B}_{1,n}(t)} \prod_{k=2}^{K} \exp(\log(\mathbf{q}_{k-1}^{-1}\mathbf{q}_k)\tilde{B}_{k,n}(t)), \tag{4.27}$$

with control points $(\mathbf{q}_1, \ldots, \mathbf{q}_K)$, can be constructed.

This spline is a weighted combination of spherical linear interpolations (SLERP) [65]

$$\mathbf{q}(s, \mathbf{q}_1, \mathbf{q}_2) = \mathbf{q}_1 \exp(s \log(\mathbf{q}_1^{-1}\mathbf{q}_2)) \quad s \in [0, 1], \tag{4.28}$$

between adjacent control points. Kim et al. chose to model the spline using these simpler SLERP curves because it makes differentiation of the spline very simple.

An important practical aspect is that the two quaternions $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation. To make sure that the SLERP interpolations take the shortest path, it is important that adjacent control points in the quaternion spline agree on a sign, e.g. by enforcing the inner product $\mathbf{q}_k^T \mathbf{q}_{k+1} > 0$.

4.5 Splines in $\mathbb{SE}(3)$

The *special euclidean group*, $\mathbb{SE}(3)$, is the group of all rigid transformations, or in this context: poses. Its elements are a subset of the 4×4 matrices, defined as

$$\left\{ \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \;:\; \mathbf{R} \in \mathbb{SO}(3), \mathbf{p} \in \mathbb{R}^3 \right\}. \tag{4.29}$$

The corresponding Lie algebra for $\mathbb{SE}(3)$ is denoted $\mathfrak{se}(3)$ [6], with elements

$$\{\boldsymbol{\xi}^\vee \in \mathbb{R}^{4\times4} : \boldsymbol{\xi} \in \mathbb{R}^6\}. \tag{4.30}$$

The coordinate vector $\boldsymbol{\xi}$ is the concatenation of two vectors

$$\boldsymbol{\xi} = \begin{bmatrix} \mathbf{v} \\ \mathbf{w} \end{bmatrix} \quad \mathbf{v}, \mathbf{w} \in \mathbb{R}^3, \tag{4.31}$$

where $\mathbf{w}$ represents a rotation, and $\mathbf{v}$ a translation. The "vee" operator $(\cdot)^\vee$, is used to construct the $\mathfrak{se}(3)$ element

$$\boldsymbol{\xi}^\vee = \begin{bmatrix} [\mathbf{w}]_\times & \mathbf{v} \\ \mathbf{0} & 0 \end{bmatrix}, \tag{4.32}$$

from the coordinate vector $\boldsymbol{\xi}$.

Just like in $\mathbb{SO}(3)$, the exponential map can be defined also on $\mathbb{SE}(3)$, with

$$\mathbf{T} = \exp(\boldsymbol{\xi}^\vee), \tag{4.33}$$

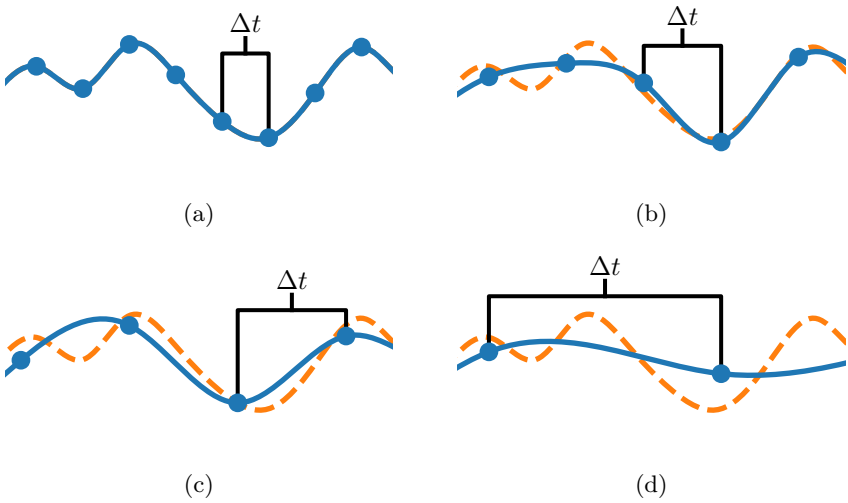


Figure 4.3: Example of how the knot spacing Δt affects the capability of the splined trajectory (solid line) to represent the true motion (dashed line). The locations of the knots are indicated by markers (dots) on the trajectory. (a) shows the optimal knot spacing, with a perfect fit. (b)-(d) show how increasing the knot spacing makes the fit worse.

and logarithm

$$\boldsymbol{\xi}^{\vee} = \log(\mathbf{T}), \tag{4.34}$$

to move between the group and the Lie algebra [67].

Lovegrove et al. [40] use the exponential map on $\mathbb{SE}(3)$ to modify the quaternion spline proposed by Kim et al. [36] and obtain the $\mathbb{SE}(3)$ spline

$$\mathbf{T}(t) = \exp(\log(\mathbf{T}_1)\tilde{B}_{1,n}) \prod_{k=2}^{K} \exp(\log(\mathbf{T}_{k-1}^{-1}\mathbf{T}_k)\tilde{B}_{k,n}(t)), \tag{4.35}$$

with control points $(\mathbf{T}_1, \ldots, \mathbf{T}_K) \in \mathbb{SE}(3)$.

4.6 Representation power

The range of motions which a spline can represent depends on the density of the knot vector. A denser spline, where knots are placed closer together, can handle a wider range of motion, as exemplified in figure 4.3. However, a too dense spline is also wasteful, and may cause overfitting.

Choosing an appropriate knot vector is therefore a crucial problem to solve before attempting to use a spline-based trajectory for continuous-time

structure from motion. Exactly how this selection should be made, and how dense the spline needs to be is, however, not immediately clear. Some cases are obvious, like only having a single valid spline segment for a trajectory that should model a long sequence with many turns. As the density of the spline is increased, it will become more and more difficult to decide when the spline is "dense enough". For example, in some applications the spline in figure 4.3c may suffice, while other applications might require the perfect model in figure 4.3a.

The model error

The assumption when solving the bundle adjustment problem in (3.10), is that the residuals are all properly weighted, using the weighting matrix, $\mathbf{W}$. This results in

$$\|\mathbf{r}\|_{\mathbf{W}}^2 = \mathbf{r}^T\mathbf{W}\mathbf{r} = \tilde{\mathbf{r}}^T\tilde{\mathbf{r}}\,, \tag{4.36}$$

where $\tilde{\mathbf{r}}$ is a whitened residual, from a standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

If the spline is dense enough to perfectly match the true motion (i.e. figure 4.3a), then the weight matrix can simply be set to the inverse covariance of the measurement noise $\mathbf{W} = \mathbf{\Sigma}^{-1}$. This is convenient, because an estimate of the measurement noise is usually available either from technical specifications of the sensor, from experience, or from an estimate on data.

When the spline is not fully able to model the true motion (i.e. figures 4.3b-4.3d), then it is obvious that the residuals will contain an additional error because of the imperfect motion model. To avoid this, the only two possibilities are to

1. make sure that the spline is always dense enough, or
2. estimate the modelling error.

Failing to do either of these will result in an unfair balancing of the measurements.

Selecting an optimal knot vector

The simplest method to select the knot vector, is to set it by hand. This is the choice made by Lovegrove et al. [40], who use uniform splines with knot spacing $\Delta t = 0.1$ seconds, and inverse noise covariance weights. If all the camera motions one wants to model have similar characteristics, finding a suitable knot spacing by hand is a valid choice. However, whenever the motion is more complex than expected, the residual balancing will be unfair.

Oth et al. [49] instead tries to find an optimal knot vector for the given data. The key insight here is that the expected value of each whitened residual

is

$$\mathbb{E}[\tilde{\mathbf{r}}^T\tilde{\mathbf{r}}] = d\,, \tag{4.37}$$

where d is the dimensionality of the residual, and that the expectation of the bundle adjustment cost function is

$$\mathbb{E}[J] = \sum_{i=1}^{N} \mathbb{E}[\tilde{\mathbf{r}}_i^T\tilde{\mathbf{r}}_i] = Nd\,. \tag{4.38}$$

Their method then works by starting from a sparse spline, which is then iteratively refined by inserting knots in spline segments where $J > \mathbb{E}[J]$.

Anderson et al. [4] also try to find an optimal trajectory model. However, they instead model the trajectory by defining its rough shape using one spline. A hierarchy of wavelet basis functions is then used to give finer details. To decide which wavelet basis functions should be used, they use the same criterion as Oth et al.

Estimating the model error

The downside of requiring a perfect trajectory model, is that it can result in very dense splines, which in turn requires more trajectory parameters to estimate. By instead estimating the model error, a sparser spline with fewer parameters can be used, and still result in balanced residuals. This also removes the need for having to iteratively find the knot vector, allowing for a faster solution.

Paper C introduces a method, called *Spline Error Weighting* (SEW), which can estimate the variance of the model error from the inertial data. SEW uses the fact that the process of fitting a spline to data can be expressed in the frequency domain, by a frequency response function $H(f)$, as described by Unser et al. [70]. In one dimension, and in the continuous domain, this means that a spline, $f(t)$, which is fit to a signal, $x(t)$, can be computed as

$$F(f) = H(f;\Delta t)\cdot X(f)\,, \tag{4.39}$$

where $F(f)$ and $X(f)$ are their respective Fourier transforms. The frequency response function $H(f)$ is of low-pass type, where the bandwidth is determined by the knot spacing, Δt. As expected, this means that an increased knot spacing (sparser spline) removes more high frequency components, resulting in a smoother spline. The connection between the spline fit error, the frequency response function, and the knot spacing, is illustrated in figure 4.4.

Using the IMU data as the signal to fit to, the variance of the model error can be computed in the frequency domain. The model error variance is then combined with the estimated measurement noise variance, and the resulting variance can then be used to weight the measurements in a balanced way. Figure 4.5 shows a synthetic example of applying SEW, and how it is a better

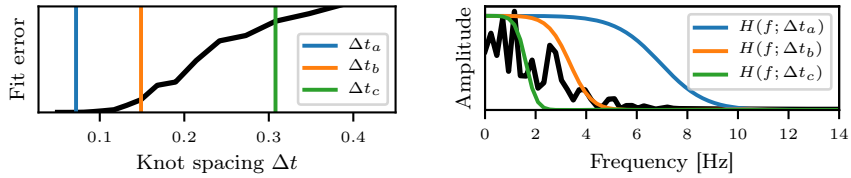


Figure 4.4: (Left) Spline fit error as a function of knot spacing. (Right) Amplitude spectrum, and examples of the frequency response function $H(f;\Delta t)$, for different choices of Δt.

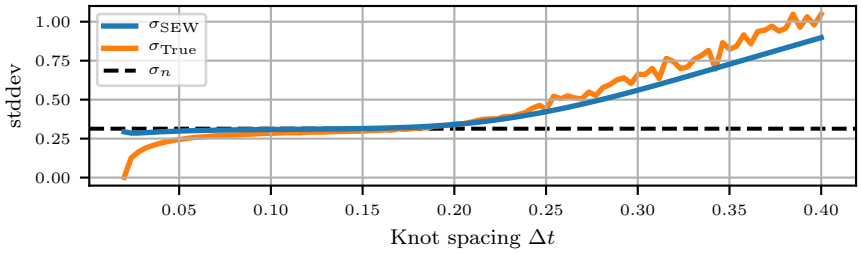


Figure 4.5: Synthetic example of application of Spline Error Weighting (SEW). SEW is a better predictor for the true residual variance, than using the measurement noise variance σ_n^2 (dashed line), for sparser splines.

estimator of the residual variance than the measurement noise weighting used in Lovegrove et al. [40].

The frequency relation in (4.39) can be used to define the quality of the spline fit, as

$$Q(\Delta t) = \frac{\text{energy after fit}}{\text{energy before fit}} = \frac{\int_{-\infty}^{\infty} |H(f;\Delta t)\cdot X(f)|^2 df}{\int_{-\infty}^{\infty} |X(f)|^2 df}. \tag{4.40}$$

In Paper C, this quality measure is used to find a suitable knot spacing. If $\hat{Q}$ is the minimum ratio of energy that should be preserved by the spline fit, then the knot spacing is the largest Δt for which $Q(\Delta t) \geq \hat{Q}$ holds.

5

Camera-IMU calibration

In order to solve the 3D reconstruction problem described in chapter 3, it is very important that all the sensors are properly calibrated. A badly calibrated sensor can otherwise introduce systematic errors, which will distort the resulting trajectory and 3D structure. In the case of visual-inertial fusion, there are two sensors: a camera and an IMU. These must not only be calibrated on their own, but must also be jointly calibrated and synchronized.

This chapter starts by outlining methods for calibrating cameras (projection model and rolling shutter readout time), and IMUs. The final part of this chapter then discusses the joint calibration and synchronization of a camera-IMU system, and more specifically the problem of finding a good initial estimate of the sought parameters.

5.1 Camera calibration

Calibration of a camera implies finding its intrinsic matrix, $\mathbf{K}$, as well as the parameters of the chosen lens distortion model, $\phi(\cdot)$, (see section 2.1).

The perhaps most common approach is the method by Zhang [71]. Here a *calibration pattern*, usually a chessboard pattern, is used to find image locations, which are in a known configuration. By exploiting the planarity of the calibration pattern, an initial estimate of the camera parameters, as well as the camera poses, can be estimated. This initial estimate is then refined by minimizing the reprojection errors, using non-linear least squares optimization.

5.2 Rolling shutter image readout calibration

The image readout time, r (see section 2.1), defines the amount of rolling shutter distortion. Although the value varies between cameras, it is constrained

to $r \in [0, T_f]$, where $1/T_f$ is the frame rate. A value of $r = 0$ is equivalent to a global shutter camera.

Oth et al. [49] finds the image readout by imaging a chessboard pattern, and reconstructing a continuous-time trajectory, with r as an additional parameter to estimate. For extra robustness, the covariance of the image residuals is modelled to take the image readout time, and motion, into account.

Jia and Evans [30] finds the image readout (and other camera-IMU parameters) by using an EKF, where the calibration parameters are added to the state. The calibration is complete when the parameters have converged. Instead of image reprojection errors, they use a rolling shutter version of the coplanarity constraint [37].

The method chosen for calibrating the rolling shutter cameras used in this work is the one by Geyer et al. [23]. Here, an LED is placed in front of the camera, and is set to flash at a known frequency. The flashing LED and the rolling shutter results in an image where bands of rows are either exposed to the flash, or not. The width of the bands in this image can then be used, together with the known flash frequency, to estimate the readout. This method gives high accuracy, but requires special equipment (i.e. the LED and a signal generator).

5.3 IMU calibration

In the context of this work, the only IMU parameters that need to be calibrated for are the accelerometer and gyroscope biases, and the measurement noise variances. The biases vary slowly, and can be modelled e.g. as constants [40], as sparse splines [20], or be included in a filter state [30, 32, 39].

Noise variances can either be taken from the technical specification of the sensor, or be estimated from data where the sensor is stationary.

5.4 Camera-IMU calibration and synchronization

When a system is built that contains multiple sensors, it is not enough to calibrate each sensor on its own. In addition, one must also find the pose transformation that relates their coordinate frames, as well as their time relation.

Relative pose

As the visual-inertial bundle adjustment problem is formulated in (3.10), it is assumed that both the IMU and camera share the same coordinate frame. This coordinate frame is simply the coordinate frame defined by the trajectory, $\mathbf{T}(t)$. However, since the camera and IMU are manufactured as two separate devices that are then joined together, their coordinate frames will not be exactly identical. Even if the manufacturing process is precise enough to exactly

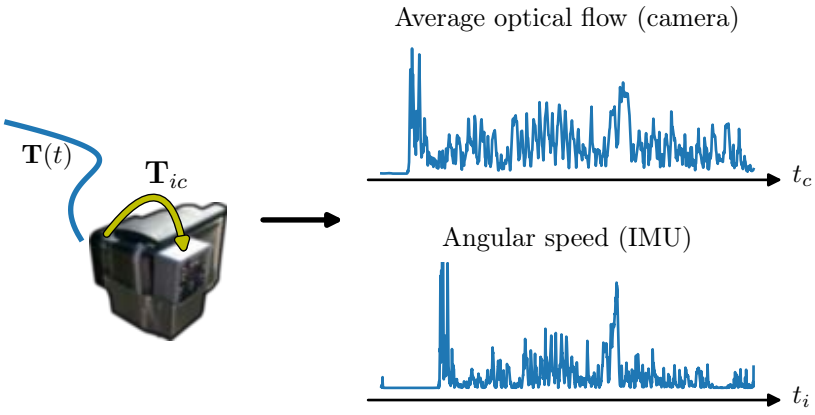


Figure 5.1: A motion $\mathbf{T}(t)$ of the camera-IMU system (left) generates measurements that can be used for calibration (right). The depicted camera-IMU system contains a GoPro Hero 3+ Black, on which a custom-made IMU logger is attached using duct tape. The coordinate frames of the two sensors are related via the relative pose, $\mathbf{T}_{ic}$.

align the orientations of the two coordinate frames, their points of origin can never be identical, as the camera origin is the aperture. Therefore, to use a camera-IMU system, the camera to IMU coordinate frame transformation

$$\mathbf{T}_{ic} = (\mathbf{R}_{ic}, \mathbf{p}_{ic}), \tag{5.1}$$

must be calibrated for.

For each sensor, s, we can define a transformation $\mathbf{T}_{ts}$ that maps points from the sensor to the trajectory coordinate frame. The relative pose can therefore be expressed as $\mathbf{T}_{ic} = \mathbf{T}_{ti}^{-1}\mathbf{T}_{tc}$ (see figure 5.1, left). In the case when there is only one IMU, it makes sense to identify its coordinate frame with the trajectory coordinate frame, which gives $\mathbf{T}_{ti} = \mathbf{I}$, and $\mathbf{T}_{ic} = \mathbf{T}_{tc}$. The visual residuals in the visual-inertial bundle adjustment cost function (3.10) are therefore changed to

$$\mathbf{y}_{k,n} - \pi(\mathbf{T}_{tc}^{-1}\mathbf{T}^{-1}(t_{k,n})\mathbf{x}_k), \tag{5.2}$$

while keeping the inertial residuals the same.

Time synchronization

Every measurement produced by the sensors has an associated *timestamp*. In some camera-IMU systems, both sensors are using the same clock to keep time, which means that their timestamps are given in the same reference time

frame. However, in many cases, the two sensors have their own clocks, which means that their timestamps are not given in the same reference time frame. Assuming that the clocks are stable, the time relation between the camera and IMU can be expressed as

$$t_c = st_i + d\,, \tag{5.3}$$

where s is a scale factor, and d is an offset. The scale factor is useful if the timestamps are not given in the same units (e.g. seconds), or if the quality of the clock of one sensor is unreliable. For example, the Microsoft Kinect used in Paper A only provides a raw, unit-less, counter as timestamp.

It should also be noted that, even in the case when the two sensors share the same clock, it is not certain that the synchronization is perfect. In theory, the timestamp should be the time instance when the measurement is recorded by the sensor, but this is not necessarily true in practice. For example, the timestamp could be recorded by the host system when the measurement is received, e.g. over USB [44]. Or, in the case of an image, the timestamp could be either the start-of-frame or end-of-frame time. Because of this uncertainty with what the timestamp really represents, performing synchronization is always a necessary activity.

Synchronization and calibration methods

To calibrate the relative pose, $\mathbf{T}_{ic}$, and find the time relation (s and d), the two sensors are subjected to a motion sequence, which is measurable in both sensors (i.e. not too large or too small). According to an observability study by Kelly and Sukhatme [32], this motion should contain rotations and accelerations around at least two axes each. The general idea is then to find the relative pose and time parameters which best explain these measurements, by using e.g. WNLS [40, 49, 31], or filtering [39, 30, 32]. One possible solution is to minimize the cost function

$$\begin{aligned} J(\mathcal{T}, \mathcal{X}, \mathbf{T}_{ic}, s, d) = &\sum_{\mathbf{y}_{k,n} \in \mathcal{Y}} \underbrace{\|\mathbf{y}_{k,n} - \pi(\mathbf{T}_{ic}^{-1}\mathbf{T}^{-1}(t_{k,n})\mathbf{x}_k)\|^2_{\mathbf{W}_v}}_{\text{Visual}} \\ &+ \sum_m \underbrace{\|\boldsymbol{\omega}_m - \nabla_\omega \mathbf{T}(st_m + d)\|^2_{\mathbf{W}_g}}_{\text{Gyroscope}} \\ &+ \sum_l \underbrace{\|\mathbf{a}_l - \nabla_a^2 \mathbf{T}(st_l + d)\|^2_{\mathbf{W}_a}}_{\text{Accelerometer}}\,, \end{aligned} \tag{5.4}$$

which is a modified version of the visual-inertial bundle adjustment problem in (3.10). Here, the IMU and trajectory coordinate frames are the same, and the camera defines the common time frame.

Regardless of which approach is chosen to find the parameters, it is necessary to find good initial estimates of both the relative pose, and the time

parameters. Otherwise the solution might not converge to the true parameters. The following sections describe methods of finding an initial estimate of the time offset and relative pose. While these can be used also as the final estimate, they are better suited as e.g. the starting point to minimize (5.4) using WNLS.

If possible, the image measurements are collected by imaging a *calibration target*, e.g. the chessboard pattern described in section 5.1. This results in an absence of outlier measurements, as well as the possibility for a simplified solution, due to the known 3D configuration of the image measurements. However, the use of a calibration target must be decided on beforehand. This inconvenience is what motivated the calibration method in Paper B, which allows calibration on a general scene, and thus does not require a calibration target.

Initial estimate of the time parameters

To find the time synchronization, when the offset is large, one approach (e.g. [44], Paper A, and Paper B) is to use correlation. By summarizing the measurements of each sensor to some 1D functions $f_c(t)$ and $f_i(t)$, which are believed to correlate, the time offset is

$$d = \arg\max_{\tau} \int f_c(t) f_i(t+\tau) dt \,. \tag{5.5}$$

In practice, this correlation is performed on measurement samples, which means that (5.5) is computed in the discrete domain, where the measurements have been resampled to the same sample rate. Since rotation is easily measured by both the camera and IMU, a prime candidate for f_i and f_c is to use angular speed (see figure 5.1). Angular speed is the norm of the angular velocity, which is directly measured by the IMU. The camera can approximate angular speed by finding relative rotations between image viewpoints. In Paper A, average optical flow magnitude is used instead of angular speed (see figure 5.1), which is faster to compute, but less accurate.

Using correlation to find the offset requires that the time scale factor, s, is already known. If that is not the case, e.g. as with the raw counters provided by the Kinect in Paper A, it has to be estimated. Since the time scale factor can be assumed to be constant, it is necessary to compute it only once for the camera-IMU system. By identifying two corresponding events $((t_{i,1}, t_{c,1}), (t_{i,2}, t_{c,2}))$ in the measurements, the scale factor can easily be computed as

$$s = \frac{t_{c,2} - t_{c,1}}{t_{i,2} - t_{i,1}} \,. \tag{5.6}$$

The accuracy of the scale estimate increases with the time difference between the two events, and by the number of measurements made.

Initial estimate of the relative pose

Unless the camera and IMU are positioned very far from each other, finding the relative orientation, $\mathbf{R}_{ic}$, is more important than finding the relative position, $\mathbf{p}_{ic}$. One approach to find the relative orientation [44], which was used in Paper A, and Paper B, is to solve

$$\mathbf{R}_{ic} = \arg\min_{\mathbf{R}} \sum_{j} \left\| \mathbf{R}\mathbf{n}_{c,j} - \mathbf{n}_{i,j} \right\|, \tag{5.7}$$

where $\mathbf{n}_{c,j}$ and $\mathbf{n}_{i,j}$ are corresponding rotation axes as measured in the camera and IMU coordinate frames, respectively. The rotation axes can be easily computed from image pairs, using essential matrix estimation (see section 3.8), and trivially computed from the IMU by integration of the gyroscope measurements.

6

Evaluation methods

Evaluating the result of a 3D-reconstruction algorithm is in theory simple: just compare the estimated camera pose trajectory, and 3D structure, with their true values. In practice, this is not so simple, and there are multiple methods for evaluation, which are more or less applicable for different methods and contexts. This chapter presents a sample of evaluation methods, and describes their use.

6.1 Ground-truth trajectories

Motion capture

Motion capture systems are well-known in the visual effects industry where they are used to apply the motions of a human actor to a computer-animated model. Most motion capture systems require that markers, e.g. flat stickers or small spheres, are placed on the body whose motion should be captured. The markers are then visible from a large set of cameras which are positioned in a known configuration in the motion capture studio. If a marker is visible in more than one camera, its position can be estimated.

Motion capture can be very accurate, with position errors in the order of a few millimeters, and have high sample rates. This makes them ideal for evaluating the quality of an estimated trajectory. However, motion capture requires all evaluations to take place in a motion capture studio, which is usually located indoors. Motion capture is therefore impractical for evaluation of systems intended to be used outdoors, or for very large scenes.

Marker-based positioning

In motion capture, markers placed on the moving body are captured by cameras. This can be reversed, by instead placing markers in the scene, which are

captured by a camera attached to the moving body. In this case, the markers need to be placed such that a sufficient number of them are always seen by the camera. While large scenes may require hundreds of markers, it is still much cheaper than covering the same area with cameras. However, the exact positions of all markers need to be determined, which can be very time consuming. An example of a dataset that employs marker-based positioning is the PennCOSYVIO [60] dataset which consists of a walking scenario captured by both normal, and RGB-D cameras, as well as inertial sensors. One problem with marker-based positioning is that it is constrained to motions where a certain amount of markers are present in each image, and can be accurately measured. This not only excludes scenarios with motion blur, but also those where the camera sometimes observes parts of the scene where markers can not be placed, e.g. when looking up in the sky.

GPS

The global positioning system (GPS), is used all over the world for navigation. The accuracy of GPS is in the order of a few meters, which is good enough when navigating by car, but not for the motions involved in handheld rolling shutter video. GPS can be augmented by differential GPS, or real-time kinematics, which improves the position accuracy by adding a set of base stations with known positions. This approach is used in the well-known KITTI dataset [22]. These systems work well for determining ground truth for larger vehicles, but their size and setup can be prohibiting for handheld scenarios.

Synthetic data

The only reasonable way to get perfect ground truth is to synthetically create it, e.g. by using one of the continuous-time trajectory models presented in section 3.3. The ground truth trajectory can then be used to generate camera and IMU measurements, by applying measurement models for these sensors. Synthetic data should preferably be generated from trajectory and sensor models that are different from those used by the algorithm under test, in order to validate the general applicability of the method.

IMU measurements depend only on the ground truth trajectory and the IMU measurement model (i.e. ∇_ω and ∇_a^2 in (3.10)), and are thus quite simple to generate. The camera measurements depend on the trajectory and camera models, but also on the 3D structure. Perfect ground truth would require realistic renderings of a synthetic 3D scene which could then be used by feature trackers to produce the camera measurements. Currently, rendering these types of realistic scenes is expensive and impractical. A simpler approach is to skip the generation of images, and instead directly generate image measurements from a set of known 3D landmarks [51].

Figure 6.1: Example of scale error calculation. (Left) and (middle) show video frames where the endpoints of the object (a door) have been connected with a red line. (Right) shows the author measuring the width of the object using a measuring tape.

6.2 Indirect trajectory quality metrics

If ground truth trajectories are not available, there is still the option to use some kind of proxy measurement as a representation of estimation quality. While these indirect metrics are not as accurate as measuring the deviation from the ground truth, they can still be useful to rank the results from different methods. Since such proxy metrics are weaker than direct methods of evaluation, a sound strategy is to use more than one.

Scale error

The use of inertial measurements results in a reconstructed trajectory that, ideally, has the same scale as the real world. The deviation in scale can therefore be used as an indication of trajectory quality.

Finding the scale of the reconstruction is a matter of measuring the length of an object, both in the real world, and in the reconstruction. To find the length of the object in the reconstruction, its endpoints are triangulated using the reconstructed trajectory, and the 3D distance between them is then computed. To find the length of the object in the real world one typically uses a measuring tape. An example of image endpoints, and the real world measurement, is shown in figure 6.1.

Endpoint error

In Paper C, the dataset was recorded such that the camera was returned to the starting point, forming a closed loop. An indicator bar mounted on a tripod was used to ensure that the start and end positions were as close as possible (i.e. within a few centimeters). The quality of a reconstructed trajectory can then be estimated by measuring how close its starting and endpoint are. It is important that the algorithms under test do not employ loop closure, because in that case a low endpoint error is expected even if the rest of the trajectory is bad.

6.3 Measuring 3D distortion

In Paper A the question was whether the rolling shutter of an RGB-D camera has any impact on the result of 3D reconstruction, using the method by Newcombe et al. [48]. When the camera is moving in a panning motion, the rolling shutter results in images where objects appear slanted. The hypothesis is that this slant would carry over to the reconstructed 3D model, where it can be measured and used as a quality value. A similar method was used for a tilting motion, except now the rolling shutter results in objects which are either contracted or elongated along the vertical axis. The slant and contraction can be measured by finding the corners of a cuboid (e.g. a box) in the 3D model.

7

Concluding remarks

This thesis has presented methods which are connected to the problem of visual-inertial fusion. Initially, the primary motivation for using an IMU was to compensate for the electronic rolling shutter in the cameras. This was then used to provide better depth images for 3D reconstruction, and also for video stabilization. However, the more recent works hint at other, more general, benefits of adding an IMU to the 3D reconstruction problem, i.e. the possibility of a correctly scaled reconstruction, as well as a reduced need for initialization.

Even though more and more camera devices are already equipped with an IMU, the need for additional calibration and synchronization will still be important to achieve the best possible results, e.g. in the case of using rolling shutter measurements. This is especially true for methods which are free of calibration targets, or special procedures, like the ones presented in this thesis.

Splines have so far proved popular for modelling continuous-time trajectories, and this is likely to remain true also in the future. For more challenging types of motion, the Spline Error Weighting (SEW) method presented in this thesis is crucial for fair balancing of the visual and inertial measurements. Also, instead of making ad-hoc selections of the type of spline, and approach for rolling shutter projection, researchers can now use the experimental results presented in this thesis to make a more informed choice.

The area of continuous-time structure from motion is still quite young, and is not very well-known. With the increased availability of inertial sensors, and the possibility of performing 3D reconstruction also on more challenging data, this will hopefully improve in the future. Future work on the SEW method includes improvements in how to handle the effect of gravity on the accelerometer measurements. Another obvious improvement is to instead use non-uniform splines, for better efficiency when dealing with motions of mixed complexity. In this work, the 3D reconstructions have been done in batch, as is the usual case in structure from motion with bundle adjustment. Continuous-

time trajectories, with splines, have previously been used also in an online, incremental, context. Adapting the methods in this thesis for the online case would also be an interesting research topic.

Recently, a new type of visual sensor known as event cameras have become available. These cameras can be seen as having an extreme form of rolling shutter, with very high temporal resolution. In the future, as these sensors gain in popularity, modelling trajectories in continuous-time will become even more relevant.

Bibliography

[1] Cenek Albl, Zuzana Kukelova, and Tomas Pajdla. "R6P - Rolling shutter absolute pose problem". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 07-12-June (2015), pp. 2292–2300. ISSN: 10636919. DOI: 10.1109/CVPR.2015.7298842.

[2] Cenek Albl, Akihiro Sugimoto, and Tomas Pajdla. "Degeneracies in Rolling Shutter SfM". In: *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016, pp. 36–51. ISBN: 978-3-319-46454-1.

[3] Sean Anderson and Timothy D. Barfoot. "Full STEAM ahead: Exactly sparse Gaussian process regression for batch continuous-time trajectory estimation on SE(3)". In: *IEEE International Conference on Intelligent Robots and Systems* 2015-Decem.3 (2015), pp. 157–164. ISSN: 21530866. DOI: 10.1109/IROS.2015.7353368.

[4] Sean Anderson, Frank Dellaert, and Timothy D. Barfoot. "A hierarchical wavelet decomposition for continuous-time SLAM". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 373–380. DOI: 10.1109/ICRA.2014.6906884.

[5] Sean Anderson, Kirk Mactavish, and Timothy D Barfoot. "Relative continuous-time SLAM". In: *The International Journal of Robotics Research* 34.12 (2015), pp. 1453–1479. ISSN: 0278-3649. DOI: 10.1177/0278364915589642.

[6] Timothy D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017. DOI: 10.1017/9781316671528.

[7] Charles Bibby and Ian Reid. "A hybrid SLAM representation for dynamic marine environments". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2010), pp. 257–264. ISSN: 10504729. DOI: 10.1109/ROBOT.2010.5509262.

[8] Michael Bosse and Robert Zlot. "Continuous 3D scan-matching with a spinning 2D laser". In: *2009 IEEE International Conference on Robotics and Automation* (2009), pp. 4312–4319. ISSN: 1050-4729. DOI: 10.1109/ROBOT.2009.5152851.

[9] Richard P. Brent. "Algorithms for Minimization without Derivatives". In: Englewood Cliffs, NJ: Prentice-Hall, 1973. Chap. 4: An Algorithm with Guaranteed Convergence for Finding a Zero of a Function.

[10] Yuchao Dai, Hongdong Li, and Laurent Kneip. "Rolling Shutter Camera Relative Pose: Generalized Epipolar Geometry". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016, pp. 4132–4140. DOI: 10.1109/CVPR.2016.448.

[11] Carl De Boor. *A practical guide to splines; rev. ed.* Applied mathematical sciences. Berlin: Springer, 2001.

[12] Frédéric Devernay and Olivier Faugeras. "Straight lines have to be straight: Automatic calibration and removal of distortion from scenes of structured environments". In: *Machine Vision and Applications* 13.1 (2001), pp. 14–24.

[13] Gregory Dudek and Michael Jenkin. "Inertial Sensors, GPS, and Odometry". In: *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 477–490. ISBN: 978-3-540-30301-5. DOI: 10.1007/978-3-540-30301-5_21.

[14] Hugh Durrant-Whyte and Tim Bailey. "Simultaneous localization and mapping: part I". In: *IEEE Robotics & Automation Magazine* 13.2 (June 2006), pp. 99–110. ISSN: 1070-9932. DOI: 10.1109/MRA.2006.1638022.

[15] Chris Engels, Henrik Stewenius, and David Nistér. "Bundle Adjustment Rules". In: *Photogrammetric Computer Vision*. Sept. 2006.

[16] Anders Eriksson, Carl Olsson, Fredrik Kahl, and Tat-Jun Chin. "Rotation Averaging and Strong Duality". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[17] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692.

[18] Per-Erik Forssén and Erik Ringaby. "Rectifying rolling shutter video from hand-held devices". In: *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society. San Francisco, USA: IEEE, June 2010. ISBN: 978-1-4244-7028-0.

[19] 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 (RSS'15)*. Rome, Italy, July 2015.

[20] Paul Furgale, Chi Hay Tong, Timothy D. Barfoot, and Gabe Sibley. "Continuous-time batch trajectory estimation using temporal basis functions". In: *The International Journal of Robotics Research* 34.14 (2015), pp. 1688–1710. ISSN: 0278-3649. DOI: 10.1177/0278364915585860.

[21] Abbas El Gamal and Helmy Eltoukhy. "CMOS Image Sensors". In: *IEEE Circuits and Devices Magazine* (May 2005).

[22] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[23] Christopher Geyer, Marci Meingast, and Shankar Sastry. "Geometric Models of Rolling-Shutter Cameras". In: *6th OmniVis WS*. 2005.

[24] Fredrik Gustafsson. *Statistical Sensor Fusion*. Studentlitteratur AB, 2012. ISBN: 978-91-44-07732-1.

[25] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, 2004. ISBN: 0521540518.

[26] Johan Hedborg, Per-Erik Forssén, and Michael Felsberg. "Fast and Accurate Structure and Motion Estimation". In: *International Symposium on Visual Computing*. Vol. 5875. Lecture Notes in Computer Science. Springer, Nov. 2009, pp. 211–222. ISBN: 978-3-642-10330-8.

[27] Johan Hedborg, Per-Erik Forssén, Michael Felsberg, and Erik Ringaby. "Rolling Shutter Bundle Adjustment". In: *IEEE Conference on Computer Vision and Pattern Recognition*. June 2012.

[28] Johan Hedborg and Björn Johansson. *Real time camera ego-motion compensation and lens undistortion on GPU*. Tech. rep. Department of Electrical Engineering, Linköping University, Sweden, 2007.

[29] Peter J. Huber. "Robust Estimation of a Location Parameter". In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101. ISSN: 0003-4851. DOI: 10.1214/aoms/1177703732.

[30] Chao Jia and Brian L. Evans. "Online Calibration and Synchronization of Cellphone Camera and Gyroscope". In: *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. Dec. 2013.

[31] Alexandre Karpenko, David Jacobs, J Baek, and Marc Levoy. "Digital video stabilization and rolling shutter correction using gyroscopes". In: *Stanford University Computer Science Tech Report CSTR* (2011).

[32] Jonathan Kelly and Gaurav S. Sukhatme. "Visual-Inertial Sensor Fusion: Localization, Mapping and Sensor-to-Sensor Self-calibration". In: *The International Journal of Robotics Research* 30.1 (2011), pp. 56–79. ISSN: 0278-3649. DOI: 10.1177/0278364910382802.

[33] Christian Kerl, Jörg Stückler, and Daniel Cremers. "Dense Continuous-Time Tracking and Mapping with Rolling Shutter RGB-D Cameras". In: *IEEE International Conference on Computer Vision (ICCV15)*. 2015.

[34] Christian Kerl, Jürgen Sturm, and Daniel Cremers. "Dense visual SLAM for RGB-D cameras". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 2100–2106. DOI: 10.1109/IROS.2013.6696650.

[35] Myoung-Jun Kim, Myung-Soo Kim, and Sung Yong Shin. "A Compact Differential Formula for the First Derivative of a Unit Quaternion Curve". In: *The Journal of Visualization and Computer Animation* 7.1 (1996), pp. 43–57.

[36] Myoung-Jun Kim, Myung-Soo Kim, and Sung Yong Shin. "A General Construction Scheme for Unit Quaternion Curves with Simple High Order Derivatives". In: *SIGGRAPH'95*. 1995, pp. 369–376.

[37] Laurent Kneip, Roland Siegwart, and Marc Pollefeys. "Finding the Exact Rotation between Two Images Independently of the Translation". In: *Computer Vision – ECCV 2012*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 696–709. ISBN: 978-3-642-33783-3.

[38] Felix Järemo Lawin, Per-Erik Forssén, and Hannes Ovrén. "Efficient Multi-Frequency Phase Unwrapping using Kernel Density Estimation". In: *European Conference on Computer Vision (ECCV)*. Amsterdam: Springer International Publishing AG, Oct. 2016.

[39] Mingyang Li, Hongsheng Yu, Xing Zheng, and Anastasios I. Mourikis. "High-fidelity sensor modeling and self-calibration in vision-aided inertial navigation". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2014), pp. 409–416. ISSN: 10504729. DOI: 10.1109/ICRA.2014.6906889.

[40] 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: *British Machine Vision Conference (BMVC)*. BMVA, Sept. 2013.

[41] David G. Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[42] Bruce D. Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'81. Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679.

[43] Kaj Madsen, Hans Bruun Nielsen, and Ole Tingleff. *Methods for Non-Linear Least Squares Problems (2nd ed.)* Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2004.

[44] Elmar Mair, Michael Fleps, Michael Suppa, and Darius Burschka. "Spatio-temporal initialisation for IMU to camera registration". In: *IEEE Int. Conf. Robot. Biomimetics*. 2011.

[45] Daniel Martinec and Tomas Pajdla. "Robust Rotation and Translation Estimation in Multiview Reconstruction". In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. June 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383115.

[46] Elias Mueggler, Guillermo Gallego, and Davide Scaramuzza. "Continuous-Time Trajectory Estimation for Event-based Vision Sensors". In: *Proceedings of Robotics: Science and Systems*. Rome, Italy, July 2015. DOI: 10.15607/RSS.2015.XI.036.

[47] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipiulation*. CRC Press, 1994.

[48] Richard A. Newcombe, Sharam Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. "KinectFusion: Real-Time Dense Surface Mapping and Tracking". In: *IEEE International Symposium on Mixed and Augmented Reality ISMAR'11*. Basel, Switzerland, Oct. 2011.

[49] Luc Oth, Paul Furgale, Laurent Kneip, and Roland Siegwart. "Rolling Shutter Camera Calibration". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR13)*. Portland, Oregon, June 2013, pp. 1360–1367.

[50] Hannes Ovrén and Per-Erik Forssén. "Camera-IMU Calibration with Robust Initialisation". In: *Proceedings of SSBA 2015 IAPR*. Non-reviewed workshop. IAPR. SSBA, Mar. 2015.

[51] Hannes Ovrén and Per-Erik Forssén. "Ground Truth for Rolling Shutter Visual-Inertial SLAM and Camera-IMU Calibration". In: *Proceedings of SSBA 2016 IAPR*. Non-reviewed workshop. IAPR. SSBA, Mar. 2016.

[52] Hannes Ovrén and Per-Erik Forssén. "Gyroscope-based video stabilisation with auto-calibration". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, WA: IEEE, May 2015, pp. 2090–2097. ISBN: 978-1-4799-6923-4. DOI: 10.1109/ICRA.2015.7139474.

[53] Hannes Ovrén and Per-Erik Forssén. "Inertial-aided Continuous-Time Structure from Motion in Practice". In: *Proceedings of SSBA 2018 IAPR*. Non-reviewed workshop. IAPR. SSBA, Mar. 2018.

[54] Hannes Ovrén and Per-Erik Forssén. "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, June 2018.

[55] Hannes Ovrén and Per-Erik Forssén. "Trajectory Representation and Landmark Projection for Continuous-Time Structure from Motion". In: *International Journal of Robotics Research* XXX.XXX (2018). Under review.

[56] Hannes Ovrén, Per-Erik Forssén, and David Törnqvist. "Better 3D with Gyroscopes". In: *Proceedings of SSBA 2013 IAPR*. Non-reviewed workshop. IAPR. SSBA, Mar. 2013.

[57] Hannes Ovrén, Per-Erik Forssén, and David Törnqvist. "Improving RGB-D Scene Reconstruction Using Rolling Shutter Rectification". In: *New Development in Robot Vision*. 2014, pp. 55–71. ISBN: 978-3-662-43858-9. DOI: 10.1007/978-3-662-43859-6_4.

[58] Hannes Ovrén, Per-Erik Forssén, and David Törnqvist. "Why Would I Want a Gyroscope on my RGB-D Sensor?" In: *Proceedings of IEEE Winter Vision Meetings, Workshop on Robot Vision (WoRV13)*. Clearwater, FL, USA: IEEE, Jan. 2013.

[59] Erik Ringaby Per-Erik Forssén and Johan Hedborg. *Computer Vision on Rolling Shutter Cameras*. CVPR 2012 Tutorial. 2012. URL: http://www.cvl.isy.liu.se/education/tutorials/rolling-shutter-tutorial/.

[60] Bernd Pfrommer, Nitin Sanket, Kostas Daniilidis, and Jonas Cleveland. "PennCOSYVIO: A challenging Visual Inertial Odometry benchmark". In: *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*. 2017, pp. 3847–3854. DOI: 10.1109/ICRA.2017.7989443.

[61] Kaihuai Qin. "General matrix representations for B-splines". In: *The Visual Computer* 16.3 (May 2000), pp. 177–186. ISSN: 1432-2315. DOI: 10.1007/s003710050206.

[62] Edward Rosten, Reid Porter, and Tom Drummond. "Faster and Better: A Machine Learning Approach to Corner Detection". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 32.1 (Jan. 2010).

[63] John Sell and Patrick O'Connor. "The Xbox One System on a Chip and Kinect Sensor". In: *IEEE Micro* 34.2 (Mar. 2014), pp. 44–53. ISSN: 0272-1732. DOI: 10.1109/MM.2014.9.

[64] Jianbo Shi and C. Tomasi. "Good features to track". In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. June 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[65] Ken Shoemake. “Animating rotation with quaternion curves”. In: *ACM SIGGRAPH computer graphics* 19.3 (1985), pp. 245–254. ISSN: 00978930. DOI: 10.1145/325334.325242.

[66] Jamie Shotton, Andrew Fitzgibbon, Andrew Blake, Alex Kipman, Mark Finocchio, Bob Moore, and Toby Sharp. “Real-Time Human Pose Recognition in Parts from a Single Depth Image”. In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2011.

[67] Hauke Strasdat. “Local Accuracy and Global Consistency for Efficient Visual SLAM”. PhD thesis. London: Imperial College of London, Department of Computing, Oct. 2012.

[68] Chi Hay Tong, Paul Furgale, and Timothy D. Barfoot. “Gaussian Process Gauss-Newton for non-parametric simultaneous localization and mapping”. In: *International Journal of Robotics Research* 32.5 (2013), pp. 507–525. ISSN: 02783649. DOI: 10.1177/0278364913478672.

[69] Bill Triggs, Philip Mclauchlan, Richard Hartley, and Andrew Fitzgibbon. “Bundle adjustment – a modern synthesis”. In: *Vision Algorithms: Theory and Practice, LNCS*. Springer Verlag, 2000, pp. 298–375.

[70] Michael Unser, Akram Aldroubi, and Murray Eden. “B-spline signal processing. II. Efficiency design and applications”. In: *IEEE Transactions on Signal Processing* 41.2 (1993), pp. 834–848. ISSN: 1053587X. DOI: 10.1109/78.193221.

[71] Zhengyou Zhang. “A Flexible New Technique for Camera Calibration”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334.

[72] Zhengyou Zhang. “Microsoft Kinect Sensor and Its Effect”. In: *IEEE MultiMedia* 19.2 (Feb. 2012), pp. 4–10. ISSN: 1070-986X. DOI: 10.1109/MMUL.2012.24.

[73] Zhengyou Zhang. “Parameter Estimation Techniques: A Tutorial With application to Conic Fitting”. In: *Journal of Image and Vision Computing* 15.1 (1997), pp. 59–76.

Papers

The papers associated with this thesis have been removed for copyright reasons. For more details about these see:

http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-148766