Linköping Studies in Science and Technology
Dissertation No. 1615

Geometric Models for Rolling-shutter and Push-broom Sensors

Erik Ringaby

Linköping University
INSTITUTE OF TECHNOLOGY

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

Linköping August 2014

Geometric Models for Rolling-shutter and Push-broom Sensors

Department of Electrical Engineering
Linköping University
SE-581 83 Linköping
Sweden

ISBN 978-91-7519-255-0 ISSN 0345-7524

Linköping Studies in Science and Technology
Dissertation No. 1615

Abstract

Almost all cell-phones and camcorders sold today are equipped with a CMOS (Complementary Metal Oxide Semiconductor) image sensor and there is also a general trend to incorporate CMOS sensors in other types of cameras. The CMOS sensor has many advantages over the more conventional CCD (Charge-Coupled Device) sensor such as lower power consumption, cheaper manufacturing and the potential for on-chip processing. Nearly all CMOS sensors make use of what is called a *rolling shutter readout*. Unlike a *global shutter readout*, which images all the pixels at the same time, a rolling-shutter exposes the image row-by-row. If a mechanical shutter is not used this will lead to geometric distortions in the image when either the camera or the objects in the scene are moving. Smaller cameras, like those in cell-phones, do not have mechanical shutters and systems that do have them will not use them when recording video. The result will look wobbly (jello effect), skewed or otherwise strange and this is often not desirable. In addition, many computer vision algorithms assume that the camera used has a global shutter and will break down if the distortions are too severe.

In airborne remote sensing it is common to use push-broom sensors. These sensors exhibit a similar kind of distortion as that of a rolling-shutter camera, due to the motion of the aircraft. If the acquired images are to be registered to maps or other images, the distortions need to be suppressed.

The main contributions in this thesis are the development of the three-dimensional models for rolling-shutter distortion correction. Previous attempts modelled the distortions as taking place in the image plane, and we have shown that our techniques give better results for hand-held camera motions. The basic idea is to estimate the camera motion, not only between frames, but also the motion during frame capture. The motion is estimated using image correspondences and with these a non-linear optimisation problem is formulated and solved. All rows in the rolling-shutter image are imaged at different times, and when the motion is known, each row can be transformed to its rectified position. The same is true when using depth sensors such as the Microsoft Kinect, and the thesis describes how to estimate its 3D motion and how to rectify 3D point clouds.

In the thesis it has also been explored how to use similar techniques as for the rolling-shutter case, to correct push-broom images. When a transformation has been found, the images need to be resampled to a regular grid in order to be visualised. This can be done in many ways and different methods have been tested and adapted to the push-broom setup.

In addition to rolling-shutter distortions, hand-held footage often has shaky camera motion. It is possible to do efficient video stabilisation in combination with the rectification using rotation smoothing. Apart from these distortions, motion blur is a big problem for hand-held photography. The images will be blurry due to the camera motion and also noisy if taken in low light conditions. One of the contributions in the thesis is a method which uses gyroscope measurements and feature tracking to combine several images, taken with a smartphone, into one resulting image with less blur and noise. This enables the user to take photos which would have otherwise required a tripod.

Populärvetenskaplig sammanfattning

Nästan alla mobiltelefoner och videokameror som säljs idag är utrustade med en CMOS-bildsensor (Complementary Metal Oxide Semiconductor) och det finns även en allmän trend att använda CMOS-sensorer i andra typer av kameror. Sensorn har många fördelar jämfört med den mer konventionella CCD-sensorn (Charge-Coupled Device) såsom lägre strömförbrukning, billigare tillverkning och möjligheten att utföra beräkningar på chippet. CMOS-sensorer i konsumentprodukter använder sig av vad som kallas en *rullande slutare*. Till skillnad från en *global slutare*, där alla pixlar avbildas samtidigt, så exponerar en sensor med rullande slutare bilden rad för rad. Kameror som använder rullande slutare kan liknas vid en skanner som läser av ett papper rad för rad. Om man rör på pappret under tiden det skannas in så kommer den slutgiltiga bilden att bli böjd eller vågig, istället för rak som originalbilden. På samma sätt kommer bilder och videor tagna med en rullande slutare att uppvisa geometriska distorsioner (förvrängningar) om antingen föremålen som filmas rör sig, eller om kameran själv flyttas. En mekanisk slutare avhjälper problemet, men dessa används inte vid videoinspelning och mindre kameror, såsom de i mobiltelefoner, har ingen mekanisk slutare alls. Avhandlingen har fokuserat på metoder för att hantera de geometriska distorsioner som uppkommer när kameran rör sig under exponering, främst genom handhållen fotografering och videoinspelning. Många datorseendealgoritmer antar att den kamera som används har en global slutare och kommer därför inte att fungera om distorsionen är för stor, men med tekniker från denna avhandling blir det lättare för forskare och konsumenter att använda kameror med rullande slutare.

De viktigaste bidragen i denna avhandling är nya tredimensionella modeller för korrigering av distorsioner från rullande slutare. Tidigare metoder modellerade distorsionerna i bildplanet och vi har visat att vår teknik ger bättre resultat för handhållna kamerarörelser. Den grundläggande idén är att uppskatta kamerans rörelse, inte bara mellan bilder i en videosekvens, utan också den rörelse som sker under tiden en enskild bild tas. Rörelsen kan skattas med hjälp av matchning av punkter mellan bilderna och genom att använda dessa kan ett matematiskt problem formuleras och lösas. Alla rader i en bild tagen med rullande slutare avbildas vid olika tidpunkter och när rörelsen för kameran är känd kan varje rad flyttas till dess korrekta position.

Microsoft Kinect är ett tillbehör till Xbox 360 som registrerar människors rörelser och tillhandahåller förutom färgbilder även bilder innehållandes avstånd mellan sensorn och föremål i rummet. Tack vare möjligheten att erhålla avståndsbilder, tillsammans med det låga priset har sensorn blivit populär att använda i datorseendesystem och på robotplattformar och om dessa är mobila kommer sensorns rörelse att ge upphov till distorsioner både i färgbilder och i avståndsbilder på grund av användningen av rullande slutare. I avhandlingen beskrivs hur man tar hänsyn till detta genom skattning av sensorns 3D-rörelse med efterföljande korrektion av 3D-punkter.

I luftburen fjärranalys är det vanligt att använda push-broomsensorer. Dessa sensorer uppvisar en liknande typ av förvrängning som för en kamera med rullande slutare, på grund av rörelsen hos flygplanet. I avhandlingen undersöks hur man

använder liknande tekniker som i fallet med rullande slutare för att rätta till push-broombilder och även olika metoder för att visualisera de korrigerade bilderna.

Förutom distorsioner uppkomna på grund av rullande slutare så har handhållna videoupptagningar ofta skakig kamerarörelse. Avhandlingen beskriver hur man gör effektiv videostabilisering, i kombination med borttagning av de geometriska distorsionerna. Utöver dessa distorsioner så är rörelseoskärpa ett stort problem vid handhållen fotografering. Bilderna blir suddiga på grund av att den som tar bilderna inte kan hålla kameran stilla och bilderna blir även brusiga om de är tagna i dåliga ljusförhållanden. Ett av bidragen i avhandlingen är en metod, som med hjälp av gyroskopmätningar och matchning av bildpunkter kombinerar flera bilder tagna med en mobiltelefon till en slutgiltig bild med både mindre brus och rörelseoskärpa. Detta medför att användaren kan ta bilder som annars skulle kräva att ett stativ används.

Acknowledgments

These past years have been really enjoyable and I would like to thank all the members of the Computer Vision Laboratory for contributing both to my research and for creating an inspiring atmosphere. Especially I would like to thank:

- Per-Erik Forssén for being an excellent supervisor, never-ending source of ideas and always having the time for discussions.
- Michael Felsberg for sharing his knowledge and allowing me to join the research group.
- Johan Hedborg for many interesting discussions regarding research and programming issues, and for convincing me to be a PhD student during my master thesis work.
- Marcus Wallenberg for, besides many research discussions, also reawakening my musical interest.
- Per-Erik Forssén, Marcus Wallenberg and Vasileios Zografos for proofreading parts of the manuscript.

Many people outside the group have also contributed and made my life busy and truly enjoyable. I would like to give some extra thanks to:

- My ninja buddies in the Bujinkan dojo for many years of training and fun both on and outside the mat.
- IK NocOut.se for creating a very social and fun training environment where I have got to know a lot of nice people.
- My clubbing friends who make the music even more pleasurable at the events.
- Jocke Holm and Johan Beckman for keeping my mind off work and for having interesting discussions of important and unimportant stuff in life.
- My mother for life long support, interest in trying to understand what I am doing and who knew, even before me I did, that I would pursue a PhD.

The research leading to this thesis has received funding from CENIIT through the Virtual Global Shutters for CMOS Cameras project.

Erik Ringaby August 2014

Contents

Part I

Background

Chapter 1

Introduction

1.1 Motivation

Almost all cell-phones and camcorders sold today are equipped with a CMOS (Complementary Metal Oxide Semiconductor) image sensor and there is also a general trend to incorporate CMOS sensors in other types of cameras. The sensor has many advantages over the more conventional CCD (Charge-Coupled Device) sensor such as lower power consumption, cheaper manufacturing and the potential for on-chip processing. Nearly all CMOS sensors make use of what is called a *rolling shutter readout*. Unlike a *global shutter readout*, which images all the pixels at the same time, a rolling-shutter camera exposes the image row-by-row. If a mechanical shutter is not used this will lead to geometric distortions in the image when either the camera or the objects in the scene are moving. Smaller cameras, like those in cell-phones, do not have mechanical shutters and systems which do have them will not use them when recording video. Figure 1.1 shows some examples of different distortions. The top left shows skew caused by a panning motion, the top right shows distortions caused by a 3D rotation and the bottom left shows distortions from a fast moving object (note that the car and the wheels are distorted differently). Almost all computer vision algorithms assume that the camera used has a global shutter. The work in this thesis will enable people to also use rolling-shutter cameras and is focused on distortions caused by camera motion, e.g. top row in figure 1.1.

In airborne remote sensing it is common to use push-broom sensors. These sensors exhibit a similar kind of distortion as a rolling-shutter camera, due to the motion of the aircraft, see figure 1.1 bottom right for an example. If the acquired images are to be registered with maps or other images, the distortions need to be suppressed. In this thesis it has been explored how to use similar techniques as for the rolling-shutter case in order to correct push-broom images.

The work leading to this thesis was conducted within the *Virtual Global Shutters for CMOS Cameras* project, and papers D and E in collaboration with the Swedish Defence Research Agency (FOI).

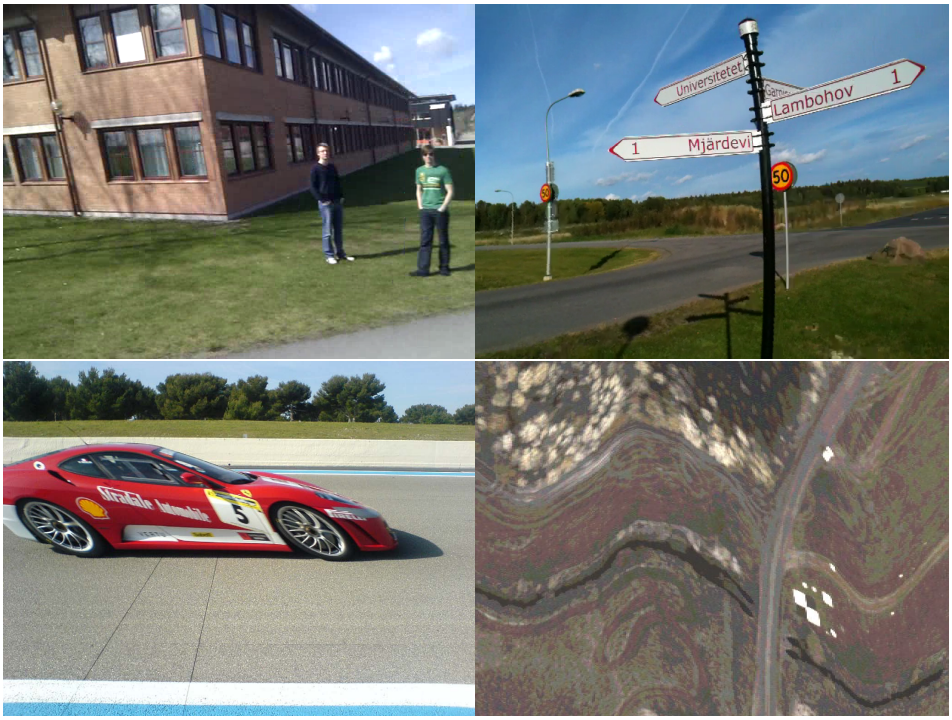


Figure 1.1: Geometric distortions in images. Top left: slanted house due to camera pan. Top right: bent pole due to camera 3D rotation. Bottom left: slanted car and curved wheels due to fast object motion. Bottom right: curved path in push-broom image due to aircraft motion.

1.2 Outline

The thesis is divided into two parts. The first part gives a background to the theory and sensors used in my work. The second part consists of six publications covering rolling-shutter and push-broom distortions.

1.2.1 Outline Part I: Background

The background part starts with chapter 2 which describes the sensors used in the publications. Chapter 3 introduces the camera models. Chapter 4 describes sensor motion estimation and how to correct for geometric distortions together with the application of video stabilisation and stacking. Chapter 5 describes the evaluation measures used, and how the ground-truth dataset was generated. The first part ends with chapter 6, concluding remarks.

1.2.2 Outline Part II: Included Publications

Preprint versions of six publications are included in Part II. The full details and abstracts of these papers, together with statements of the contributions made by the authors, are given below.

Paper A: Rectifying rolling shutter video from hand-held devices

> Per-Erik Forssén and Erik Ringaby. Rectifying rolling shutter video from hand-held devices. In *IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco, USA, 2010. IEEE Computer Society.

Abstract:
This paper presents a method for rectifying video sequences from *rolling shutter* (RS) cameras. In contrast to previous RS rectification attempts we model distortions as being caused by the 3D motion of the camera. The camera motion is parametrised as a continuous curve, with knots at the last row of each frame. Curve parameters are solved for using non-linear least squares over inter-frame correspondences obtained from a KLT tracker. We have generated synthetic RS sequences with associated ground-truth to allow controlled evaluation. Using these sequences, we demonstrate that our algorithm improves over to two previously published methods. The RS dataset is available on the web to allow comparison with other methods.
Contribution:
This paper was the first to correct rolling-shutter distortions by modelling the 3D camera motion. It also introduced the first rolling-shutter dataset. The author contributed to the rotation motion model, produced the dataset, and conducted the experiments.

Paper B: Efficient Video Rectification and Stabilisation for Cell-Phones

> Erik Ringaby and Per-Erik Forssén. Efficient video rectification and stabilisation for cell-phones. *International Journal of Computer Vision*, 96(3):335–352, 2012.

Abstract:
This article presents a method for rectifying and stabilising video from cell-phones with *rolling shutter* (RS) cameras. Due to size constraints, cell-phone cameras have constant, or near constant focal length, making them an ideal application for calibrated projective geometry. In contrast to previous RS rectification attempts that model distortions in the image plane, we model the 3D rotation of the camera. We parameterise the camera rotation as a continuous curve, with knots distributed across a short frame interval. Curve parameters are found using non-linear least squares over inter-frame correspondences from a KLT tracker. By smoothing a sequence of reference rotations from the estimated curve, we can at a small extra cost, obtain a high-quality image stabilisation. Using synthetic RS sequences with associated ground-truth, we demonstrate that our rectification improves over two other methods. We also compare our video stabilisation with the methods in iMovie and Deshaker.
Contribution:
This paper extends paper A, by allowing camera motions that are non constant during a frame capture, a new GPU-based forward interpolation, and the application of video stabilisation. The author was the main source of the findings for the importance of spline knot positions, the GPU based interpolation, and implemented the stabilisation.

Paper C: Scan Rectification for Structured Light Range Sensors with Rolling Shutters

> Erik Ringaby and Per-Erik Forssén. Scan rectification for structured light range sensors with rolling shutters. In *IEEE International Conference on Computer Vision*, Barcelona, Spain, November 2011. IEEE Computer Society

Abstract:
Structured light range sensors, such as the Microsoft Kinect, have recently become popular as perception devices for computer vision and robotic systems. These sensors use CMOS imaging chips with electronic rolling shutters (ERS). When using such a sensor on a moving platform, both the image, and the depth map, will exhibit geometric distortions. We introduce an algorithm that can suppress such distortions, by rectifying the 3D point clouds from the range sensor. This is done by first estimating the time continuous 3D camera trajectory, and then transforming the 3D points to where they would have been, if the camera had been stationary. To ensure that image and range data are synchronous, the camera trajectory is computed from KLT tracks on the structured-light frames, after suppressing the structured-light pattern. We evaluate our rectification, by measuring angles

between the visible sides of a cube, before and after rectification. We also measure how much better the 3D point clouds can be aligned after rectification. The obtained improvement is also related to the actual rotational velocity, measured using a MEMS gyroscope.
Contribution: This paper was the first to address the rolling-shutter problem on range scan sensors. Compared to paper A and paper B, the cost function is defined on 3D features, and the full 6 DOF motion can be estimated and corrected for. The author contributed to the motion estimation, feature rejection steps, and the experiments.

Paper D: Co-alignment of Aerial Push-Broom Strips using Trajectory Smoothness Constraints

> Erik Ringaby, Jörgen Ahlberg, Per-Erik Forssén, and Niclas Wadströmer. Co-alignment of aerial push-broom strips using trajectory smoothness constraints. In *Proceedings SSBA'10 Symposium on Image Analysis*, pages 63–66, March 2010

Abstract:
We study the problem of registering a sequence of scan lines (a *strip*) from an airborne push-broom imager to another sequence partly covering the same area. Such a registration has to compensate for deformations caused by attitude and speed changes in the aircraft. The registration is challenging, as both strips contain such deformations.

Our algorithm estimates the 3D rotation of the camera for each scan line, by parametrising it as a linear spline with a number of knots evenly distributed in one of the strips. The rotations are estimated from correspondences between strips of the same area. Once the rotations are known, they can be compensated for, and each line of pixels can be transformed such that the ground trace of the two strips are registered with respect to each other.
Contribution: This paper explored the possibility of using the previously introduced rolling-shutter correction scheme to register push-broom strips, by using smoothness constraints. The author contributed to the registration and conducted the experiments.

Paper E: Anisotropic Scattered Data Interpolation for Pushbroom Image Rectification

> Erik Ringaby, Ola Friman, Per-Erik Forssén, Thomas Opsahl, Trym Vegard Haavardsholm, and Ingebjørg Kåsen. Anisotropic scattered data interpolation for pushbroom image rectification. *IEEE Transactions in Image Processing*, 2014

Abstract:
This article deals with fast and accurate visualization of pushbroom image data from airborne and spaceborne platforms. A pushbroom sensor acquires images in a line-scanning fashion, and this results in scattered input data that needs

to be resampled onto a uniform grid for geometrically correct visualization. To this end, we model the anisotropic spatial dependence structure caused by the acquisition process. Several methods for scattered data interpolation are then adapted to handle the induced anisotropic metric and compared for the pushbroom image rectification problem. A trick that exploits the semi-ordered line structure of pushbroom data to improve the computational complexity several orders of magnitude is also presented.
Contribution: This paper models the spatial dependence structure of pushbroom data and is shown to be anisotropic. Five methods for scattered data interpolation are extended to handle the anisotropic nature of pushbroom data and compared for the image rectification problem. The author contributed to the extension of the forward interpolation method, the surface structure model and conducted the experiments.

Paper F: A Virtual Tripod for Hand-held Video Stacking on Smartphones

> Erik Ringaby and Per-Erik Forssén. A virtual tripod for hand-held video stacking on smartphones. In *IEEE International Conference on Computational Photography*, Santa Clara, USA, May 2014. IEEE Computer Society

Abstract:
We propose an algorithm that can capture sharp, low-noise images in low-light conditions on a hand-held smartphone. We make use of the recent ability to acquire bursts of high resolution images on high-end models such as the iPhone5s. Frames are aligned, or stacked, using rolling shutter correction, based on motion estimated from the built-in gyro sensors and image feature tracking. After stacking, the images may be combined, using e.g. averaging to produce a sharp, low-noise photo. We have tested the algorithm on a variety of different scenes, using several different smartphones. We compare our method to denoising, direct stacking, as well as a global-shutter based stacking, with favourable results.
Contribution: This paper explores the possibility to use gyroscope measurements to reduce rolling-shutter artifacts and register several images in order to create an image stack, resulting in a low-noise sharp image. The author contributed to the implementation of the iOS data collection application, gyroscope bias and gyroscope/frame synchronisation optimisation, translation model and conducted the experiments.

Other Publications

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

> Gustav Hanning, Nicklas Forslöw, Per-Erik Forssén, Erik Ringaby, David Törnqvist, and Jonas Callmer. Stabilizing cell phone video using inertial measurement sensors. In *The Second IEEE International Workshop on Mobile Vision*, Barcelona, Spain, November 2011. IEEE.

Johan Hedborg, Erik Ringaby, Per-Erik Forssén, and Michael Felsberg. Structure and motion estimation from rolling shutter video. In *The Second IEEE International Workshop on Mobile Vision*, Barcelona, Spain, November 2011.

Johan Hedborg, Per-Erik Forssén, Michael Felsberg, and Erik Ringaby. Rolling shutter bundle adjustment. In *IEEE Conference on Computer Vision and Pattern Recognition*, Providence, Rhode Island, USA, June 2012. IEEE Computer Society.

Erik Ringaby, Jörgen Ahlberg, Niclas Wadströmer, and Per-Erik Forssén. Co-aligning aerial hyperspectral push-broom strips for change detection. In *Proceedings of SPIE Security+Defence*, volume 7835, Tolouse, France, September 2010. SPIE, SPIE Digital Library.

Chapter 2

Sensors

All imaging sensors used in this thesis share the property of sequential acquisition of an image frame. How the sensors work will be described in the following sections.

2.1 Rolling-shutter sensors

The function of a camera shutter is to allow light to pass through for a determined period of time. The shutter used can either be mechanical or electronic and have a global, block or rolling exposure method. In a global-shutter camera, all pixels in a frame are imaged at a single time instance. Rolling shutter on the other hand is a technique used when acquiring images by scanning the frame. Instead of imaging the scene at a single time instance, the image rows are sequentially reset and read out. The rows which are not being read out continue to be exposed. Figure 2.1 shows the difference between image integration with a global-shutter and rolling-shutter camera. The rolling-shutter method has the advantage of longer integration times, as shown in the bottom figure, which increases the sensitivity.

The two most common image sensors used in digital cameras are the CCD (Charge-Coupled Device) and the CMOS (Complementary Metal Oxide Semiconductor) image sensors. Generally, CCD sensors use global shutters and CMOS use rolling shutters. There are CMOS sensors with a global shutter, where all the pixels are exposed to light at the same time and at the end of integration time they are transferred to a light-shielded storage area simultaneously. After this the signals are read out.

In addition to increased sensitivity, the CMOS sensors are also cheaper to manufacture, they use less power and it is also simple to integrate other kind of electronics on the chip. Almost all camera-equipped cell-phones make use of a rolling shutter and the CMOS sensor is gradually replacing the CCD sensor in other segments such as camcorders and video capable SLR's. The rolling shutter will however introduce distortions when the scene or camera is moving, and the amount of these distortions depend on how fast the shutter "rolls". A rule of thumb is that the higher the resolution is, the slower the sensor will be, and furthermore

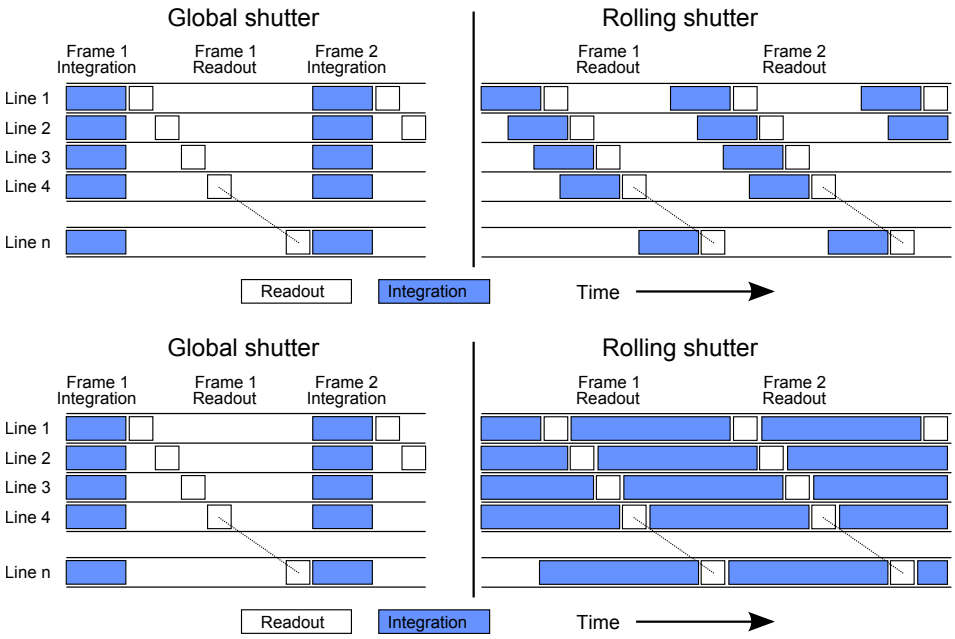


Figure 2.1: Global-shutter and rolling-shutter image integration.

expensive sensors are usually faster. Almost all computer vision algorithms assume a global-shutter camera, but techniques from this thesis allow researchers and others to also use rolling-shutter cameras.

2.2 Kinect sensor

In 2010, Microsoft released the Kinect sensor which is designed to provide motion input to the Xbox 360 gaming device. The sensor has gained popularity in the vision community due to its ability to deliver quasi-dense depth maps in 30 Hz, combined with a low price. The hardware consists of a near infrared (NIR) laser projector (A), a CMOS colour sensor (B) and a NIR CMOS sensor (C), see figure 2.2.

The laser projector is used to project a structured light pattern onto the scene. The NIR CMOS sensor images this pattern and the device uses triangulation to create a depth map. The image resolution is 640×480 when using an update of 30 Hz, but it is also possible to receive NIR and colour frames in 1280×1024 resolution. The depth map can be obtained at the same time as either the NIR image or the colour image, but the colour and NIR images cannot be obtained at the same time.

Both the NIR and colour sensors have electronic rolling shutters. Since the Kinect sensor is designed to be stationary and objects in front of it do not move

Figure 2.2: The Kinect sensor, (A) NIR laser projector, (B) CMOS colour sensor, (C) CMOS NIR sensor

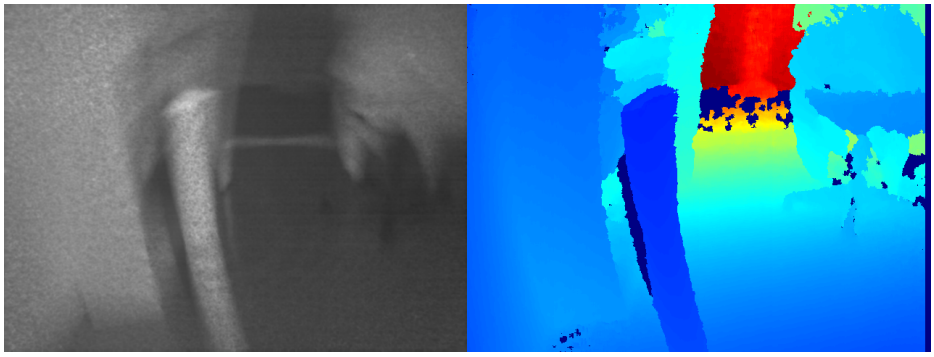


Figure 2.3: Distortions (straight pole and wall look bent) in the NIR and depth images caused by fast sensor motion.

that fast (or very close to the sensor), the rolling-shutter distortions are usually not a big problem. If on the other hand the sensor is used on a mobile platform it will have noticeable distortions, see figure 2.3 for an example of a fast rotation. The straight pole and the wall look bent due to the moving sensor. The two image sensors are not synchronised, so the same rows in the depth image and the colour image are, in general, not imaged at the same time.

2.3 Push-broom sensors

Push-broom sensors are commonly used in airborne remote sensing. The images, also called *strips* or *swaths*, from a push-broom sensor have similar geometric

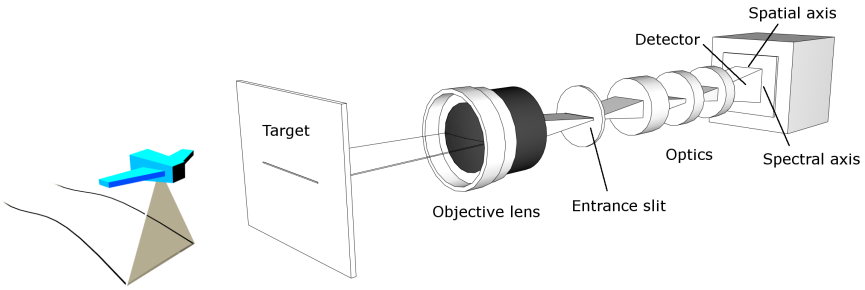


Figure 2.4: Left: How the 1D sensor "paints" the image. Right: Different spectral bands separated on the sensor using a prism.

distortions to those from a rolling-shutter sensor, but the sensors differ a great deal in their design.

Instead of capturing a two dimensional image, the sensor has a single line of pixels and "paints" the image by exploiting the ego-motion of the moving platform, see figure 2.4 left. The sensor itself is two dimensional and a prism refracts the light into different wavelengths along one of the axes of the hyper-spectral sensor (figure 2.4, right). The number of spectral bands depends on the sensor used.

If the imaging platform (e.g. aircraft) moves in a linear trajectory we would have to solve a simple problem, but this rarely the case. When the aircraft rotates, or moves away from the path, geometric distortions will be present in the image, see figure 1.1 bottom right.

There are also hyper-spectral sensors which use two spatial dimensions, but record the different wavelengths at different time steps. In this case, the registration has to be done across different spectral bands instead, but that is not considered here.

2.4 Gyroscope sensors

A gyroscope sensor measures angular velocities and is used in some of the work presented in this thesis. There exist different types of gyroscopes such as mechanical, solid-state ring lasers, fibre-optic and quantum gyroscopes. Many modern smartphones today make use of Micro-Electro-Mechanical Systems (MEMS) technology where it is common that the device includes multiple-axis gyroscope and accelerometers. A three-axis gyroscope enables the calculation of the device yaw, pitch and roll and has been used in the experiments described in paper F.

2.5 Other sensors

Other imaging sensors with similar geometry to rolling shutter, but not covered in this thesis are crossed-slits [28], and moving LIDAR[4, 3].

Chapter 3

Camera models

Some computer vision algorithms operate only in the image plane and do not care which camera has been used to record the image. In this work a model for the camera is needed, and we are using the pin-hole camera model. The following sections will describe the standard (global-shutter) model, and our rolling-shutter version. Lens distortions are not considered in this work.

3.1 Pin-hole camera with global shutter

The pin-hole camera model is a simple model which describes how 3D points in the world project onto the image plane. The camera aperture corresponds to a point and no lenses are used to describe the focusing of light. Figure 3.1 shows how a 3D object projects onto an image plane.

The relationship seen in figure 3.1 can be expressed as:

$$\frac{x}{f} = \frac{X}{Z} \tag{3.1}$$

$$\frac{y}{f} = \frac{Y}{Z}. \tag{3.2}$$

This relationship, together with a translation of the origin, skew and aspect ratio can also be described in matrix notation using homogeneous coordinates:

$$\begin{pmatrix} \lambda x \\ \lambda y \\ \lambda \end{pmatrix} = \begin{pmatrix} f & s & c_x \\ 0 & f\alpha & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3.3}$$

$$\mathbf{x} = \mathbf{K}\mathbf{X}. \tag{3.4}$$

The matrix $\mathbf{K}$ contains the *intrinsic* or *internal* camera parameters, and describes how the camera transforms the inhomogeneous point $\mathbf{X}$ onto the image. c_x and c_y describe the translation of the principal point required to move the origin

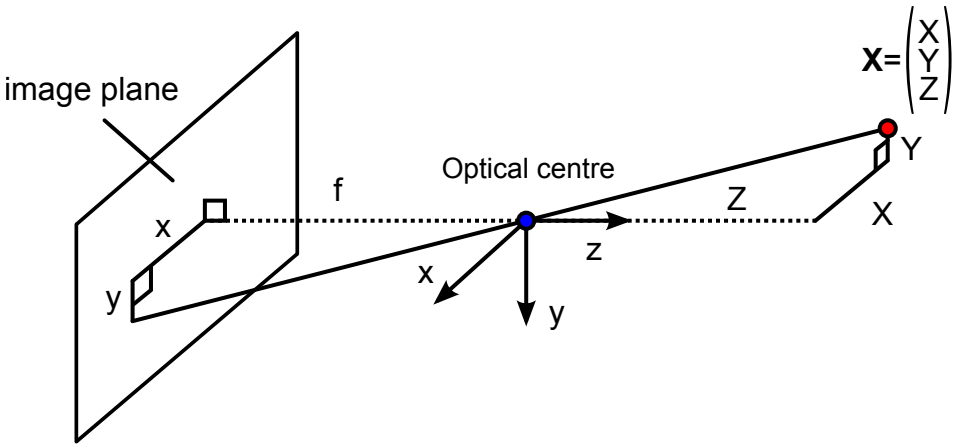


Figure 3.1: The pinhole camera model projects a 3D point $\mathbf{X}$ onto the image plane.

into image coordinates. The focal length f, in x and y direction may be different due to the aspect ratio α. The pixels may also be skewed, but in most cases $s = 0$.

Cameras used in this thesis, e.g. the one in iPhone 3GS, have a (near) constant focal length, which enables us to calibrate the camera once. We have also seen that transferring the intrinsic camera parameters between smartphone cameras of the same model works well. See section 3.4 for how the parameters are calibrated.

The *extrinsic* or *external* camera parameters describe how the camera relates to a world coordinate system. This relation, or transformation, can be described as a translation $\mathbf{d}$ and a rotation $\mathbf{R}$ and expressed as a matrix multiplication:

$$\mathbf{x} = \mathbf{K}[\mathbf{R}|\mathbf{d}]\tilde{\mathbf{X}}, \tag{3.5}$$

where $\tilde{\mathbf{X}}$ is a homogeneous point, i.e. $\tilde{\mathbf{X}} = [\mathbf{X}^T \ 1]^T$.

3.2 Pin-hole camera with rolling shutter

When a rolling-shutter camera is stationary and is imaging a rigid scene, the same model as the global-shutter case may be used. The model must however be changed when the camera is moving. The internal camera parameters are still the same (we have fixed focal lengths), but the external parameters are now time dependent. By assuming that the scanning begins at the top row, down to the bottom row we get:

$$\mathbf{x} = \mathbf{K}[\mathbf{R}(t)|\mathbf{d}(t)]\tilde{\mathbf{X}}, \tag{3.6}$$

where $t = 0$ represents the first row of the frame.

With this representation we can describe the camera's positions and orientations during a frame capture, and correct for the geometric distortions due to this motion.

3.2.1 Motion models

Instead of modelling the full camera motion as the source of the distortions one can simplify the model to three different special cases: pure rotation, pure translation and imaging of a planar scene. By choosing one of these models the estimation is simplified, which will be described in section 4.2. The pure rotation case assumes that the camera only rotates around the optical centre, which simplifies equation 3.6 to:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(t)\mathbf{X}. \tag{3.7}$$

If the camera is imaging a planar scene the motion can be described by:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(t)(\mathbf{X} + \mathbf{d}(t)) = \mathbf{K}\mathbf{R}(t)\mathbf{D}(t)\hat{\mathbf{X}}\,, \tag{3.8}$$

$$\text{where} \quad \mathbf{D} = \begin{pmatrix} 1 & 0 & d_1 \\ 0 & 1 & d_2 \\ 0 & 0 & d_3 \end{pmatrix}\,, \tag{3.9}$$

and $\hat{\mathbf{X}}$ is a three element vector containing the non-zero elements of $\mathbf{X}$, and a 1 in the third position. If the motion is a pure translation, 3.8 simplifies to:

$$\mathbf{x} = \mathbf{K}\mathbf{D}(t)\hat{\mathbf{X}}\,. \tag{3.10}$$

In paper A we came to the conclusion that the rotation model was the best for hand-held camera motions. When a user holds the camera, the main cause for the motion (and also the cause for the distortions) is rotation. If we only look at changes during a short time interval, e.g. 2-3 frames, the camera does not translate significantly. There are however notable exceptions to this, where the translation is the dominant component e.g. footage from a moving platform, such as a car.

3.3 Motion Blur

Other than rolling-shutter distortions, motion blur is a big problem for hand-held footage. Motion blur becomes apparent when something changes during the integration of the image and can be due to camera motion or moving objects, just as for the rolling-shutter case. If the exposure time is shortened, the blur will be reduced but at the same time more noise will be present. Therefore this is mostly an issue during image capture in low light conditions and is present both for cameras with global and rolling shutters.

Many methods try to estimate the camera motion during image capture, or the *blur kernel*, in order to deblur the image and obtain a sharp version. In this thesis the camera motion is estimated for several frames, which are then registered and stacked together in order to get one sharp image, called *video stacking*. Instead of using one long exposure, with resultant blurring, many short exposures are used in sequence. When the photographer has a *static aim* (i.e. tries to aim at a fixed point in space), these individual exposures tend to have blur smears in a random distribution of directions. This means that when the frames are aligned we obtain an effective *point spread function* (PSF) that is much more compact than one from a single long exposure, as can be seen in figure 3.2.

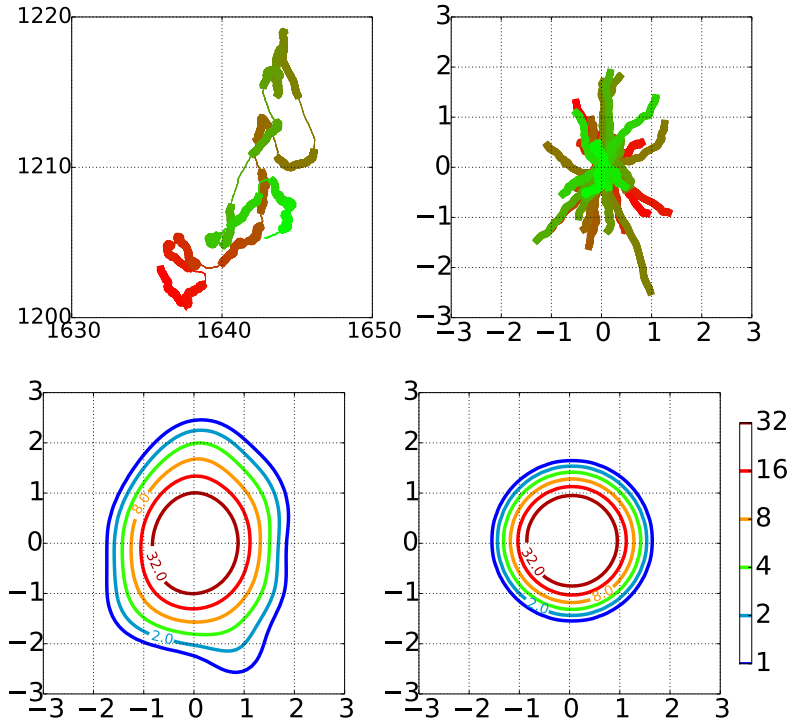


Figure 3.2: Illustration of video stacking idea. Top left: Trace of central pixel where colours indicate time, ranging from red to green. Thick segments indicate individual exposures. Top right: Alignment of the exposure segments. Bottom left: Iso contours of the effective PSF. Bottom right: Corresponding iso contours for a Gaussian with $\sigma = 0.5$.

3.4 Camera calibration

The algorithms in this thesis require calibrated cameras. We use the OpenCV implementation of Zhang's method [27] for camera calibration, which requires a number of images of a planar checkerboard pattern from different orientations. The intrinsic parameters $\mathbf{K}$, see section 3.1, are acquired this way and the lens distortion parameters are neglected.

On a rolling-shutter camera, an additional parameter also needs to be estimated, the readout time. The rolling-shutter chip frame period $1/f$ (where f is the *frame rate*) is divided into a *readout time* t_r, and an *inter-frame delay*, t_d as:

$$1/f = t_r + t_d \,. \tag{3.11}$$

Figure 3.3 shows this relation. The inter-frame delay is useful to know when the continuous camera motion is estimated. For more details on the readout time calibration, see Appendix A in paper B.

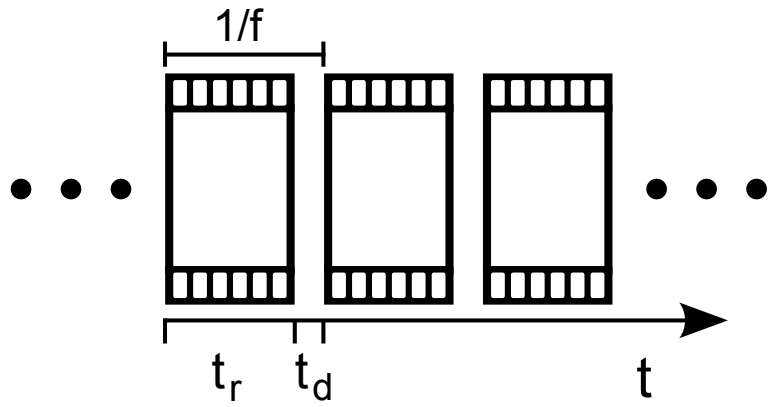


Figure 3.3: Relation between the frame period $1/f$, readout time t_r, and inter-frame delay t_d.

3.5 Push-broom model

The push-broom sensor exploits the ego-motion of the moving platform when creating the image. We do however neglect the translational component of the motion and model the distortion of a strip as a sequence of rotation homographies:

$$\mathbf{H}(t) = \mathbf{K}\mathbf{R}(t)\mathbf{K}^{-1} \,. \tag{3.12}$$

This means that we model the sensor as rotating purely about its optical centre and thus the imaged ground patch is modelled as being on the interior surface of a sphere. This will cause some distortions in the reconstruction, but if the radius of the sphere (i.e. the aircraft altitude) is large enough (compared to the strip length), this distortion is small.

Chapter 4

Geometric distortion correction

The distortions corrected for in this thesis are those caused by motion of the sensor. This is done by exploiting the continuity of the camera motion in rolling-shutter video. Feature points are detected and tracked across frames and used to estimate the camera ego-motion, or synchronisation with a gyroscope. The distortions are more severe when shooting video compared to pictures, since the user usually tries to hold the camera steady for pictures, but it is still necessary to do correction when combining several images or when high precision is needed. When depth is available, as for the Kinect sensor, the 3D points can also be used to estimate the motion.

Co-alignment of push-broom strips is a bit different since each strip comes from a single flight and we typically only have a few strips (compared to many frames in a video). Also, they might not overlap as much as two consecutive frames in a video, but within each strip the sensor has a continuous motion.

4.1 Point correspondences

For rolling-shutter video we detect points using the good features to track detector [24]. These are then tracked using the KLT-tracker [14] in order to acquire correspondences across frames. The KLT-tracker uses an image patch in one image and estimates the patch position in the next frame. It does so by using a spatial intensity gradient search which minimises the Euclidean distance between the corresponding patches. To be able to cope with large motions we use a scale pyramid approach.

We employ a cross-checking step, as in [2], which uses an additional tracking from the second image back to the first one. Only those points which return to their original position are regarded as inliers. Figure 4.1 shows points rejected using a threshold of 0.5 pixels in red and accepted points in green.

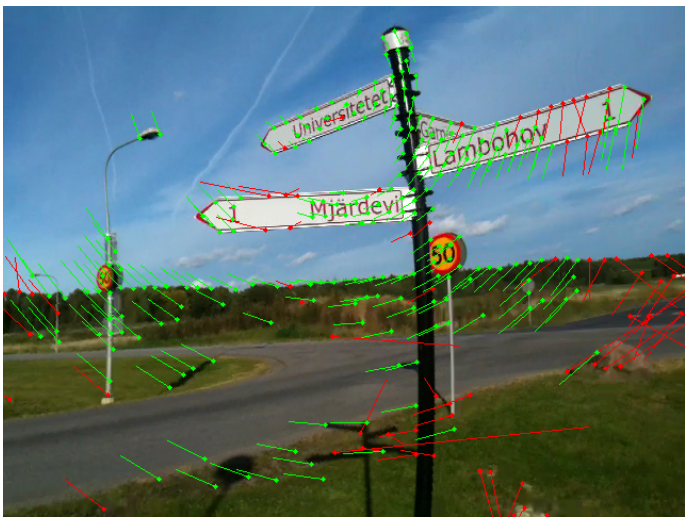


Figure 4.1: Tracked points between two frames. Rejected points in red, and accepted points in green.

Since push-broom strips are acquired at different times, tracking is difficult to do. Less overlap than between video frames and also larger changes in illumination makes feature matching a more suitable method for correspondence search than e.g. KLT. We use SIFT features [13] and match them to acquire correspondences for an initial registration of the strips.

4.2 Camera Motion estimation

The sparse point correspondences can be used to estimate the camera motion. The assumption is that the camera is moving in a static scene, so all displacement vectors are due to camera motion.

The camera motion is estimated through iterative non-linear least squares (Levenberg- Marquardt) by minimisation of the cost function associated with the camera motion model.

Since the image rows are exposed at different times, one would like to have the camera pose for each of them. This will result in a high number of parameters to be estimated and we therefore model the motion as a spline. In that way, we only estimate the parameters for a certain number of points along this curve, called *knots*. This spline exploits that the motion is smooth and interpolates all needed poses between the knots.

4.2.1 Motion parametrisation

In section 3.2 the different motion models were described and for the full model the motion is represented as a sequence of rotations and translations (the knots). The translations are represented as a three element vector and the rotation can be represented as a 3×3 matrix $\mathbf{R}$, a unit quaternion, or a three element axis-angle vector $\mathbf{n} = \phi\hat{\mathbf{n}}$. $\hat{\mathbf{n}}$ is the corresponding unit vector to $\mathbf{n}$, which defines the axis where the rotation is taking place and ϕ is the magnitude of $\mathbf{n}$ which corresponds to the rotation angle around the axis. Most of the work here make use of the axis-angle representation during the optimisation, since it is a minimal representation of a 3D rotation.

Converting from this representation to a rotation matrix is done using the matrix exponent, which for rotations simplifies to Rodrigues formula:

$$\mathbf{R} = \text{expm}(\mathbf{n}) = \mathbf{I} + [\hat{\mathbf{n}}]_x \sin\phi + [\hat{\mathbf{n}}]_x^2(1 - \cos\phi) \tag{4.1}$$

$$\text{where } [\hat{\mathbf{n}}]_x = \frac{1}{\phi}\begin{pmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{pmatrix}. \tag{4.2}$$

To convert a rotation matrix back to vector form, the matrix logarithm can be used and for rotations the following closed form exists:

$$\mathbf{n} = \text{logm}(\mathbf{R}) = \phi\hat{\mathbf{n}}\,, \quad \text{where} \quad \begin{cases} \tilde{\mathbf{n}} = \begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix} \\ \phi = \tan^{-1}(||\tilde{\mathbf{n}}||, \text{tr}\mathbf{R} - 1) \\ \hat{\mathbf{n}} = \tilde{\mathbf{n}}/||\tilde{\mathbf{n}}||\,. \end{cases} \tag{4.3}$$

Interpolation

For interpolation of translations we are using a linear interpolation:

$$\mathbf{d}_{\text{interp}} = (1 - w)\mathbf{d}_1 + w\mathbf{d}_2\,, \tag{4.4}$$

where $\mathbf{d}_1$ and $\mathbf{d}_2$ are two translation vectors (three elements) and $w \in [0, 1]$ is the weight parameter.

Interpolation of rotations is slightly more complicated due to the periodic structure of SO(3). In most of the work here we use SLERP (Spherical Linear intERPolation) [25] with an interpolation parameter $\tau \in [0, 1]$ between two knot rotations:

$$\mathbf{n}_{\text{diff}} = \text{logm}\left(\text{expm}(-\mathbf{n}_1)\text{expm}(\mathbf{n}_2)\right) \tag{4.5}$$

$$\mathbf{R}_{\text{interp}} = \text{expm}(\mathbf{n}_1)\text{expm}(\tau\mathbf{n}_{\text{diff}}). \tag{4.6}$$

$\mathbf{n}_1$ and $\mathbf{n}_2$ are two rotation axis-angle vectors and $\mathbf{R}_{\text{interp}}$ is the resulting rotation matrix.

SLERP gives constant-speed transition between two rotations and is the shortest path on the rotation manifold (geodesic). Many other splines exist for doing

the rotation interpolation and in paper F we compare SLERP, Cubic, Quartic and B-splines.

4.2.2 Optimisation

By assuming that the row which is exposed first is the top one, the row number is proportional to time. When using the rotation only model, two corresponding homogeneous image points $\mathbf{x}$, and $\mathbf{y}$ are projected from the 3D point $\mathbf{X}$ as:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(N_x)\mathbf{X}\ , \text{ and } \quad \mathbf{y} = \mathbf{K}\mathbf{R}(N_y)\mathbf{X} \tag{4.7}$$

where N_x and N_y correspond to the time parameters, e.g. the row number for point $\mathbf{x}$ and $\mathbf{y}$ respectively. This gives us the relation:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(N_x)\mathbf{R}^T(N_y)\mathbf{K}^{-1}\mathbf{y} = \mathbf{H}\mathbf{y}\,. \tag{4.8}$$

The positions of the knots are discussed in paper B. When these positions have been decided, the rotation from an arbitrary row N_{curr} (relative to the first row in the first image) is acquired by:

$$\mathbf{R} = \texttt{spline}(\mathbf{n}_m, \mathbf{n}_{m+1}, \tau)\,, \text{ for} \tag{4.9}$$

$$\tau = \frac{N_{\text{curr}} - N_m}{N_{m+1} - N_m}\,, \quad \text{where} \quad N_m \leq N_{\text{curr}} \leq N_{m+1}, \tag{4.10}$$

and N_m, N_{m+1} are the two neighbouring knot times.

The cost function to be minimised is the summed (symmetric) image-plane residuals of a set of corresponding points $\mathbf{x}_k \leftrightarrow \mathbf{y}_k$:

$$J = \sum_{k=1}^{K} d(\mathbf{x}_k, \mathbf{H}\mathbf{y}_k)^2 + d(\mathbf{y}_k, \mathbf{H}^{-1}\mathbf{x}_k)^2, \tag{4.11}$$

$$\text{where} \quad d(\mathbf{x}, \mathbf{y})^2 = (x_1/x_3 - y_1/y_3)^2 + (x_2/x_3 - y_2/y_3)^2\,. \tag{4.12}$$

Here K is the total number of correspondences between two images. It is also possible to use correspondences from more than two images in the cost function. When using the rotation only model, $\mathbf{H}$ is defined in (4.8), and here it would be beneficial to use a small number of frames per optimisation, in case the motion also includes translations. When using the planar scene model from equation 3.8, $\mathbf{H}$ is defined by:

$$\mathbf{H} = \mathbf{K}\mathbf{R}(N_x)\mathbf{D}(N_x)\mathbf{D}(N_y)^{-1}\mathbf{R}^T(N_y)\mathbf{K}^{-1}. \tag{4.13}$$

If the rotations are replaced with the identity matrix, the pure translation case is estimated instead.

Full motion estimation

If the 3D points $\mathbf{X}$ also are known, as in paper C, the cost function can be defined on these instead, resulting in estimation of the full 6 degrees-of-freedom camera

motion imaging an arbitrary scene. If $\mathbf{X}_1$ and $\mathbf{X}_2$ are two corresponding 3D points reconstructed from two different images and depth maps, they can be transformed to the position $\mathbf{X}_0$. This is the position where the reconstructed point should have been, if it was imaged at the same time as the first row in the first image:

$$\mathbf{X}_0 = \mathbf{R}(N_1)\mathbf{X}_1 + \mathbf{d}(N_1) \tag{4.14}$$

$$\mathbf{X}_0 = \mathbf{R}(N_2)\mathbf{X}_2 + \mathbf{d}(N_2). \tag{4.15}$$

By assuming that the scene is static, the difference between these points can be used to estimate the motion, resulting in the minimisation of:

$$J = \sum_{k=1}^{K} ||\mathbf{R}(N_{1,k})\mathbf{X}_{1,k} + \mathbf{d}(N_{1,k}) - \mathbf{R}(N_{2,k})\mathbf{X}_{2,k} - \mathbf{d}(N_{2,k})||^2, \tag{4.16}$$

where $N_{1,k}$ and $N_{2,k}$ are the rows where the kth 3D point is observed in the first and second image respectively.

Gyroscope-camera synchronisation

Instead of using visual features to estimate the camera motion, other sensors such as gyroscopes and accelerometers can be used. In this case it is necessary to make sure that the camera's and sensor's coordinate systems are aligned. In the thesis different smartphone devices are used where the two coordinate systems are assumed to have the same origin and a global transformation can be determined manually once for every smartphone model.

In addition to the coordinate systems it is important to have the two different inputs synchronised. The time delay between the two sources can be estimated by minimising the residuals in equation 4.11 where the points are obtained by image feature tracking. Here, instead of estimating the camera rotation at the knots, angular velocities are given at specific time stamps and can be integrated and interpolated to an orientation corresponding to the time the point where imaged. The timestamp for the gyroscope, t_{gyro}, is related to the frame timestamp t_i as in equation 4.17:

$$t_{gyro} = t_i + t_r \frac{x_2}{h} + t_{delay}, \tag{4.17}$$

where t_r is the readout time described in section 3.4, x_2 the current row for the point, h the frame hight and t_{delay} is the time delay we would like to estimate.

To improve the performance even further we use the gyroscope sample model $\mathbf{g} = \hat{\mathbf{g}} + \mathbf{b}$, where $\mathbf{g}$ is the observed sample and $\mathbf{b}$ is the gyroscope bias. The bias corrected gyroscope samples are then integrated to obtain an orientation sequence.

4.3 Image rectification

In this thesis image rectification is the process of resampling the input image to a version which looks more rigid. When the camera motion has been estimated,

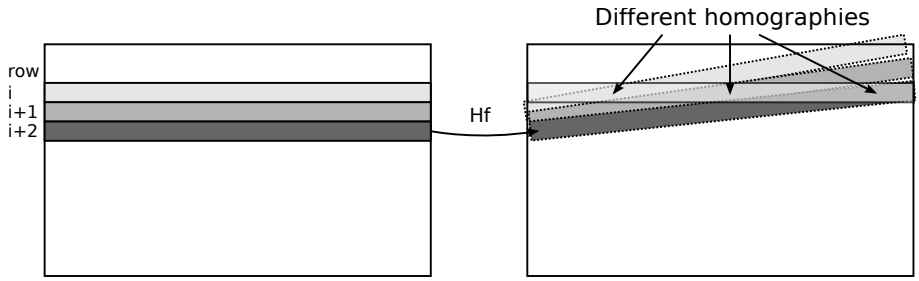


Figure 4.2: Left: Distorted input image. Right: Rectified output image.

i.e. its pose at the time instances of the knots (which corresponds to a certain image row) the poses for all the image rows can be acquired through interpolation. By using a regular grid on the input image, each row can be transformed by a different homography to create the forward mapping. The coordinate system to be transformed to can be chosen as a specific row, e.g. the one corresponding to the first or middle row of the image. This means that this *reference row* will be exactly the same in the input image and the rectified image. When using a pure rotation as motion model the rectification equation becomes:

$$\mathbf{x}' = \mathbf{K}\mathbf{R}_{\text{ref}}\mathbf{R}^T(N)\mathbf{K}^{-1}\mathbf{x}, \tag{4.18}$$

where $\mathbf{x}$ is the input image coordinate, $\mathbf{x}'$ its rectified position, $\mathbf{R}^T$ the rotation corresponding to the time instance the pixel was imaged and $\mathbf{R}_{\text{ref}}$ the rotation for chosen reference row.

If equation 4.18 is reversed, the equation for the inverse mapping becomes:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(N)\mathbf{R}_{\text{ref}}^T\mathbf{K}^{-1}\mathbf{x}'\,. \tag{4.19}$$

It is not possible to use this inverse interpolation correctly, since different pixels within a row should be transformed with different homographies, see figure 4.2. The pixels within a row in the input image do however share the same homography and can be used to correctly transform the image.

If the depth is known, the 3D points can be rectified by:

$$\mathbf{X}' = \mathbf{R}_{\text{ref}}(\mathbf{R}(N)\mathbf{X} + \mathbf{d}(N)) + \mathbf{d}_{\text{ref}}, \tag{4.20}$$

where $\mathbf{X}$ is the original distorted 3D point and $\mathbf{X}'$ is the rectified version. Also, if the depth map and video frame are to be rectified, $\mathbf{X}'$ can be projected back to the image plane and the corresponding intensity or depth value can be saved.

4.3.1 Image resampling

When the forward mapping has been calculated, the image must be resampled to a regular grid in order to be visualised, and this can be done in different ways. This

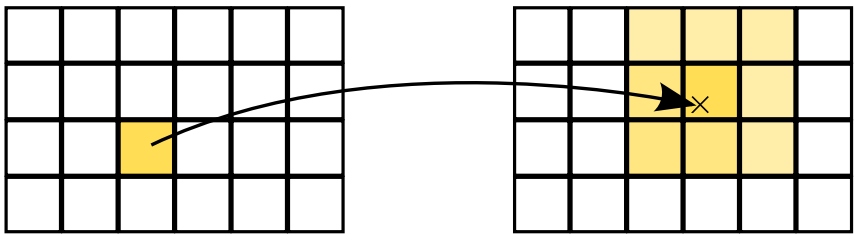


Figure 4.3: Illustration of the splatting method. In this case an input pixel (left) is smeared into a 3×3 region in the output grid (right).

scattered data interpolation can be divided into two different schemes: inverse and forward interpolation and in paper E five different methods are compared using push-broom data. An inverse interpolation means that each point in the output grid, $\mathbf{u}$, is mapped back to the input domain where the interpolation takes place by a weighted sum of the neighbouring input samples, $\mathbf{u}_i$:

$$\hat{z}(\mathbf{u}) = \sum_{\mathbf{u}_i \in \mathcal{N}(\mathbf{u})} w_i \, z\left(\mathbf{u}_i\right). \tag{4.21}$$

How the weights w_i are chosen depends on the interpolation method. In *nearest neighbour interpolation* only the nearest sample value is considered, meaning w_i will be 1 for the closest one and 0 for all other samples. Another choice is to choose the weights depending on the inverse distance to the sample as in *Inverse Distance Weighted Interpolation* [23]. Instead of using the distance to calculate the weights, *Natural Neighbors interpolation* [5] uses an area based measure by using Delaunay triangulation. *Kriging interpolation* [12] in general uses the covariance function between sample locations to derive the optimal weights in equation 4.21.

When the input data is irregularly sampled as here, one is faced with the computational problem of identifying the neighbours, and another way of doing the resampling is to do a forward interpolation, e.g. splatting. This method "smears" each input pixel into a region (e.g. 3×3 closest output grid locations), see figure 4.3, and the output RGB values $\mathbf{y}(\mathbf{u}) = (r, g, b, w)$ are updated as:

$$\mathbf{y}(\mathbf{u}) \leftarrow \mathbf{y}(\mathbf{u}) + \begin{bmatrix} w(\mathbf{u}) z(\mathbf{u}_i) \\ w(\mathbf{u}) \end{bmatrix}. \tag{4.22}$$

The weights w_i depend on the grid location and can e.g. be chosen as:

$$w(\mathbf{u}) = \exp(-.5||\mathbf{u} - \mathbf{x}'||^2/\sigma^2) \tag{4.23}$$

where σ is a smoothing parameter and $\mathbf{x}'$ is the rectified pixel location. After looping over all pixels they are normalised by the forth element, creating an output RGB image. If the camera motion is very fast, a local 3×3 region may not be enough to fill all output pixels and a larger region has to be used. For the rolling-shutter correction a fast forward mapping can be performed on a graphics

processing unit (GPU) and at the same time do the image resampling without any risk of holes. A mesh can be placed on the input image and the GPU transforms each row to their rectified position. Values between rows are automatically interpolated (in hardware) so there is no risk of holes.

Paper E examines different interpolation methods on push-broom data using an anisotropic distance measure and by also taking the surface structure into account.

4.4 Global alignment

When the sensor motion has been estimated, the rectified frames, or the tracked and rectified points can be used in other algorithms which do not take the rolling-shutter effect into account. Video stabilisation (paper B) and video stacking (paper F) can be efficiently implemented by a selection of the common coordinate system during the frame rectification.

4.4.1 Video stabilisation

The rectification technique described in section 4.3 allows for an efficient implementation of video stabilisation. When an image is rectified, all the rows are transformed to a common coordinate system corresponding to the reference row. Instead of transforming each image to e.g. the middle row, one can do a temporal smoothing of all reference rows in the image sequence and use the smoothed versions instead.

Smoothing of rotations can be achieved by matrix averaging:

$$\tilde{\mathbf{R}}_k = \sum_{l=-n}^{n} w_l \mathbf{R}_{k+l} \tag{4.24}$$

where the temporal window is $2n+1$ and w are weights for the input rotations $\mathbf{R}_k$. The output of (4.24) is not guaranteed to be a rotation matrix, but this can be enforced by constraining it to be a rotation [8]:

$$\begin{aligned} \hat{\mathbf{R}}_k &= \mathbf{U}\mathbf{S}\mathbf{V}^T, \text{ where} \\ \mathbf{U}\mathbf{D}\mathbf{V}^T &= \text{svd}(\tilde{\mathbf{R}}_k), \text{ and } \mathbf{S} = \text{diag}(1, 1, |\mathbf{U}||\mathbf{V}|). \end{aligned} \tag{4.25}$$

The motion estimation is done during a short frame interval, and since all optimisations have different origins they have to be transformed to a common coordinate system. The stabilisation will be a restriction on the orientation, and since the pure rotation model may not hold for a long video sequence there might still be some translation left, but not so much to be disturbing.

4.4.2 Video stacking

Video stacking on hand-held sequences is quite similar to video stabilisation. The biggest difference is that for stacking, all the frames should be registered to one common position. When using the rotation only motion model there might be

Figure 4.4: Zoomed in examples between global frame alignment (left) and our rolling-shutter aware method (right).

some translation between the first and the later frames, even though the user tries to hold the camera still. In order to avoid doing full structure from motion the scene can be approximated with a fronto-parallel plane when estimating the camera translation. A point $\mathbf{y}$ in one of the frames may be re-projected onto this scene plane as $\mathbf{u}$ using:

$$\mathbf{u} = \lambda \mathbf{K}^{-1}\mathbf{y} = \lambda \begin{pmatrix} u_1 & u_2 & 1 \end{pmatrix}^T . \tag{4.26}$$

A global 3D displacement, $\mathbf{d} = \begin{pmatrix} \Delta X & \Delta Y & \Delta Z \end{pmatrix}^T$, can be estimated by minimising the residuals between the re-projected point $\mathbf{x}$ in equation 4.27 and the corresponding point in the reference image.

$$\mathbf{x} = \mathbf{K}(\lambda \mathbf{K}^{-1}\mathbf{y} + \mathbf{d}) \tag{4.27}$$

The displacement can then be used during the rectification process to stack the images at the same time.

Figure 4.4 shows the difference between using the rolling-shutter aware method described here and a global (non-rolling-shutter aware) version. Even though the user has tried to hold the camera still, the rolling-shutter image capture makes the global version blurry, see paper F for details.

Chapter 5

Evaluation

This chapter describes the generated ground-truth dataset, and the methods used for evaluation of the algorithms.

5.1 Ground-truth generation

In order to do a controlled evaluation, a synthetic dataset was developed for paper A and extended in paper B. The Autodesk Maya software was used to generate different camera motions in a 3D scene. Each rolling-shutter frame was simulated by combining 480 global-shutter frames. One row from each global-shutter frame was combined to form a rolling-shutter frame, starting at the top row and sequentially moving down to the bottom row. Figure 5.1 shows different kinds of generated camera motions in the scene.

The ground-truth for rolling-shutter rectification is a global-shutter frame. Which global-shutter frame to be used depends on at which time instance (i.e. which input image row) the distorted image is to be reconstructed. Global-shutter frames corresponding to the first, middle and last row have been generated. Depending on the motion, some parts of the ground-truth frame (borders and occlusions) are not visible in the rolling-shutter frame. For this reason, visibility masks have been generated to indicate which pixels in the ground-truth frames can be reconstructed from the corresponding rolling-shutter frame.

5.2 Evaluation measures

In paper A we introduced the first rolling-shutter dataset. This enabled us to do a comparison of different settings and methods. When a rolling-shutter frame has been rectified by the algorithm it can be compared to the generated ground-truth by calculating the average Euclidean distance to the colour pixel values in the ground-truth images. In order to evaluate only the rectification, and not the methods ability to interpolate and extrapolate values the distance is only

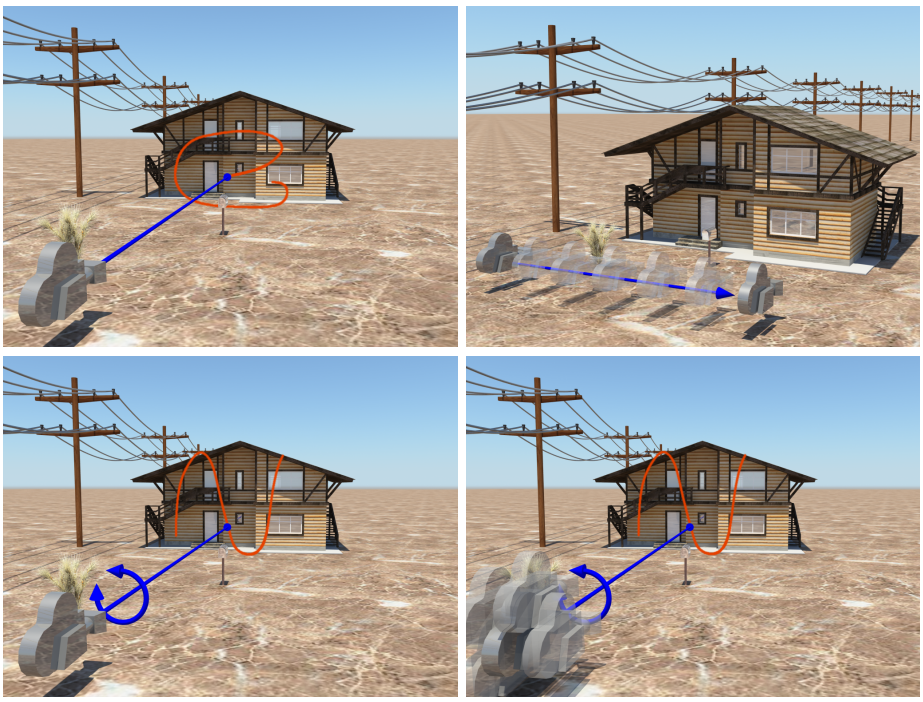


Figure 5.1: Four categories of synthetic sequences. Left to right, top to bottom: #1 rotation only, #2 translation only, # full 3DOF rotation. and #4 3DOF rotation and translation.

calculated within the valid mask. Pixels that deviate more than a certain threshold are counted as incorrect. This measure is however more sensitive in high-contrast regions, than in regions with low contrast. In paper B, we therefore used a variance-normalised error measure:

$$\epsilon(\mathbf{I}_{\text{rec}}) = \sum_{k=1}^{3} \frac{(\mu_k - I_{\text{rec},k})^2}{\sigma_k^2 + \varepsilon\mu_k^2} \,. \tag{5.1}$$

Here μ_k and σ_k are the means and standard deviations of each colour band in a small neighbourhood of the ground-truth image pixel (we use a 3×3 region), $I_{\text{rec},k}$ is the pixel value in the rectified image for colour channel k and ε is a small value that controls the amount of regularisation. This measure also has the benefit of being less sensitive to sub-pixel rectification errors.

5.2.1 Video stabilisation

Paper B also introduced an efficient method to do video stabilisation, and this is more difficult to evaluate since we both want to reduce the image plane motions and maintain a correct geometry. When no ground-truth is available, one can

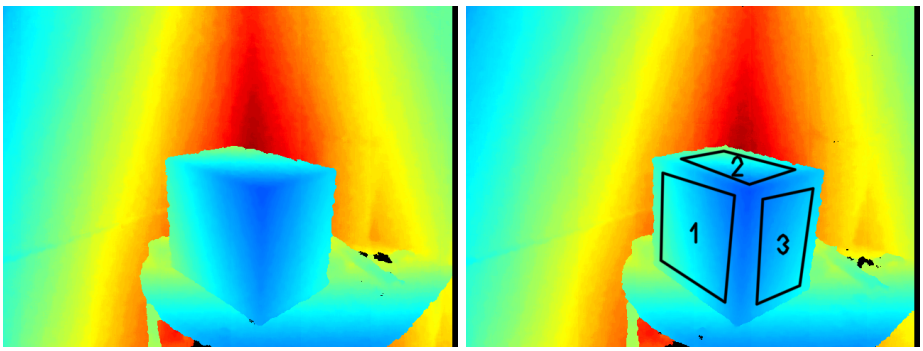


Figure 5.2: Left: Depth frame from a static sensor. Right: Manually marked planes on frame captured during sensor rotation.

evaluate image plane motion by comparing consecutive frames in a video with a certain motion. A video captured when a person is walking forward and holding the camera will be shaky but consecutive frames will be very similar if the stabilisation algorithm is good, and image plane motion from such a sequence is thus used as an evaluation measure.

Another evaluation method, used in [9], is to do a user study. Such a study was conducted as a blind experiment, where users were shown pairs of videos and asked to choose the one they thought looked the best.

5.2.2 Point cloud rectification

When evaluating the rectification of 3D point clouds, a practical method is to measure geometrical properties of a known object, e.g. comparing the angles between the visible sides of a box, before and after rectification. A ground-truth angle can be obtained by imaging the box when the sensor is stationary, see figure 5.2. The plane angles can be estimated by finding the cube normals using RANSAC [6] and computing the angle between two normals using the formula:

$$\Theta_{k,l} = \sin^{-1}(\|\hat{\mathbf{n}}_k \times \hat{\mathbf{n}}_l\|)\,, \tag{5.2}$$

where $\hat{\mathbf{n}}_k$ and $\hat{\mathbf{n}}_l$ are normal vectors for the two planes. By doing this it is possible to show that the rectified point clouds are more geometrically correct than the unrectified ones.

5.2.3 Push-broom

In papers D and E push-broom data were considered. The data in paper D did not have any ground-truth, and visual inspection was used to evaluate the registration quality as it is quite easy to observe, see figure 5.3.

Figure 5.3: Result of co-alignment of push-broom strips. Left: Overlap of two strips. Right: Difference of two strips.

In paper E, different interpolation methods were implemented and compared. The methods will predict slightly different values $\hat{z}(\mathbf{u}_i)$ and they can be compared to actual sample values $z(\mathbf{u}_i)$ in the dataset not used during the interpolation. This way, the dataset can be used to calculate the relative error which is used as the evaluation measure:

$$\varepsilon(\mathbf{p}) = \frac{1}{|\mathcal{E}|} \sum_{i \in \mathcal{E}} \frac{|\hat{z}(\mathbf{u}_i) - z(\mathbf{u}_i)|}{z(\mathbf{u}_i)}, \tag{5.3}$$

where $|\mathcal{E}|$ is the size of the evaluation set.

5.2.4 Stacking

Video stabilisation and video stacking is quite similar but with the difference that for video stabilisation, frames far from each other may differ a great deal (that is why we compare consecutive frames in section 5.2.1) while all the frames in a stack should be registered to common frame. This makes it possible to evaluate the stacking results using the standard deviation over time across a stack of frames, see equation 5.4. The standard deviation will increase either if the stacking is poor or if there are moving objects in the scene. The measure is averaged across all pixels to obtain a scalar measure:

$$\sigma_{\text{avg}} = \frac{1}{3|\Omega|} \sum_{\mathbf{x} \in \Omega} \sum_{c=1}^{3} \sqrt{\frac{1}{K} \sum_{k=1}^{K} (I_{k,c}(\mathbf{x}) - I_{\text{avg},c}(\mathbf{x}))^2}\,, \tag{5.4}$$

$$\text{where} \quad I_{\text{avg},c}(\mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} I_{k,c}(\mathbf{x})\,, \tag{5.5}$$

k is a specific frame in the stack, c the colour channel, Ω is the set of image coordinates in the frames, and $|\Omega|$ is the set size.

Another method is to use a physical tripod, taking a long exposure and use this as a ground-truth. The problem with this is that you have to do an alignment

between the stack and the ground-truth since it is difficult to image the scene from the exact same position. The scene may also have changed between the acquisition of the ground-truth and the dataset. Because of this we instead use the dataset itself to calculate the evaluation measure.

Chapter 6

Concluding remarks

This chapter summarises the main results and discusses possible areas of future work.

6.1 Results

The methods presented in this thesis can be used to increase the usability of rolling-shutter cameras, both for researchers and end users. The main contributions are the development of the three-dimensional models for rolling-shutter distortion correction. Paper A was the first paper describing this and gave superior results for hand-held camera motions compared to image-based methods. We also introduced the first rolling-shutter dataset which enables other researchers to evaluate their algorithms. Paper B introduced an efficient video stabilisation method in combination with the image rectification. A new GPU-based forward interpolation was also introduced and the paper extended the motion model to cope with faster motions.

Typically, when the Kinect sensor is used on mobile platforms it has to be moved slowly, or in a move-stop-look image acquisition so that the rolling-shutter artifacts are kept at a minimum. With the technique from paper C the data is rectified, and the sensor can be moved in an arbitrary manner.

Paper D introduced methods for co-aligning push-broom strips using similar techniques as for the rolling-shutter case, using image only data. In paper E five different interpolation methods were extended to handle the anisotropic nature of the push-broom data and compared for the image rectification problem.

Instead of using only visual measurements, paper F also explored the possibility of using gyroscope data to reduce the rolling-shutter artifacts. This was done together with a stacking procedure which combined several hand-held images into one resulting low-noise sharp image. This enables the user to take photos which would otherwise have required a physical tripod.

6.2 Future work

The image-based motion estimation assumes that the scene is stationary. During evaluation it has been shown that it is robust to some object motion in the video, but if a large part of the optical flow originates from fast-moving objects, a motion segmentation (and local rectification) may be required. In [1] they have a model for small objects with low-frequency motions but objects with high-frequency motion is more challenging.

It would also be interesting to improve the quality and the temporal resolution of the motion estimation. Possible ways may be to use a more dense optical flow, variable knot positions, to model lens distortion and to optimise over a whole sequence. This may enable the algorithm to cope with even faster camera motions, such as when it is attached to a vibrating engine.

Another interesting future work would be an auto-calibration step, since it is quite cumbersome to manually calibrate each different camera model. In [16] a calibration method is proposed which does not require specialised hardware, but still uses a calibration pattern. Paper D combined image rectification with the intention of reducing blur and image noise ([15] present a combined rolling-shutter and motion blur model, and [26] take the rolling shutter into account during the blur estimation), and it would be interesting to combine it with even more applications such as panorama stitching, augmented reality and so on.

The co-alignment of push-broom strips is currently not good enough for automatic change-detection and a more advanced motion model and possible estimation or incorporation of a height map may be required.

In [10] the 6 degrees-of-freedom motion was estimated for a monocular camera using rolling-shutter aware bundle adjustment. This made it possible to do structure from motion using a cell-phone with any kind of motion, but it is still not as stable as when using a global-shutter camera. It would be interesting to combine it with the variable knot positions from paper B and C, and the different interpolation schemes from paper F.

Bibliography

[1] Simon Baker, Eric Bennett, Sing Bing Kang, and Richard Szeliski. Removing rolling shutter wobble. In *IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco, USA, June 2010. IEEE Computer Society.

[2] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. In *IEEE ICCV*, Rio de Janeiro, Brazil, 2007.

[3] A. Banno, T. Masuda, T. Oishi, and K. Ikeuchi. Flying laser range sensor for large-scale site-modeling and its applications in bayon digital archival project. *Int. J. Comput. Vision*, 78(2-3):207–222, July 2008.

[4] Michael Bosse and Robert Zlot. Continuous 3d scan-matching with a spinning 2d laser. In *ICRA09*, Kobe, Japan, May 2009.

[5] Jean Braun and Malcolm Sambridge. A numerical method for solving partial differential equations on highly irregular evolving grids. *Nature*, 376(24):655–660, 1995.

[6] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, June 1981.

[7] Per-Erik Forssén and Erik Ringaby. Rectifying rolling shutter video from hand-held devices. In *IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco, USA, 2010. IEEE Computer Society.

[8] Claus Gramkow. On averaging rotations. *International Journal of Computer Vision*, 42(1/2):7–16, 2001.

[9] Gustav Hanning, Nicklas Forslöw, Per-Erik Forssén, Erik Ringaby, David Törnqvist, and Jonas Callmer. Stabilizing cell phone video using inertial measurement sensors. In *The Second IEEE International Workshop on Mobile Vision*, Barcelona, Spain, November 2011. IEEE.

[10] Johan Hedborg, Per-Erik Forssén, Michael Felsberg, and Erik Ringaby. Rolling shutter bundle adjustment. In *IEEE Conference on Computer Vision and Pattern Recognition*, Providence, Rhode Island, USA, June 2012. IEEE Computer Society.

[11] Johan Hedborg, Erik Ringaby, Per-Erik Forssén, and Michael Felsberg. Structure and motion estimation from rolling shutter video. In *The Second IEEE International Workshop on Mobile Vision*, Barcelona, Spain, November 2011.

[12] D. G. Krige. A statistical approach to some mine valuations and allied problems at Witwatersrand. Master's thesis, University of Witwatersrand, South Africa, 1951.

[13] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[14] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI'81*, pages 674–679, 1981.

[15] Maxime Meilland, Tom Drummond, and Andrew I. Comport. A unified rolling shutter and motion blur model for 3d visual registration. In *ICCV*, pages 2016–2023, 2013.

[16] L Oth, P T Furgale, L Kneip, and R Siegwart. Rolling shutter camera calibration. In *Proc. of The IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, Portland, USA, June 2013.

[17] Erik Ringaby, Jörgen Ahlberg, Per-Erik Forssén, and Niclas Wadströmer. Co-alignment of aerial push-broom strips using trajectory smoothness constraints. In *Proceedings SSBA'10 Symposium on Image Analysis*, pages 63–66, March 2010.

[18] Erik Ringaby, Jörgen Ahlberg, Niclas Wadströmer, and Per-Erik Forssén. Co-aligning aerial hyperspectral push-broom strips for change detection. In *Proceedings of SPIE Security+Defence*, volume 7835, Tolouse, France, September 2010. SPIE, SPIE Digital Library.

[19] Erik Ringaby and Per-Erik Forssén. Scan rectification for structured light range sensors with rolling shutters. In *IEEE International Conference on Computer Vision*, Barcelona, Spain, November 2011. IEEE Computer Society.

[20] Erik Ringaby and Per-Erik Forssén. Efficient video rectification and stabilisation for cell-phones. *International Journal of Computer Vision*, 96(3):335–352, 2012.

[21] Erik Ringaby and Per-Erik Forssén. A virtual tripod for hand-held video stacking on smartphones. In *IEEE International Conference on Computational Photography*, Santa Clara, USA, May 2014. IEEE Computer Society.

[22] Erik Ringaby, Ola Friman, Per-Erik Forssén, Thomas Opsahl, Trym Vegard Haavardsholm, and Ingebjørg Kåsen. Anisotropic scattered data interpolation for pushbroom image rectification. *IEEE Transactions in Image Processing*, 2014.

[23] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the ACM National Conference*, 1968.

[24] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR'94*, Seattle, June 1994.

[25] Ken Shoemake. Animating rotation with quaternion curves. In *Int. Conf. on CGIT*, pages 245–254, 1985.

[26] Ondrej Sindelar, Filip Sroubek, and Peyman Milanfar. Space-variant image deblurring on smartphones using inertial sensors. June 2014.

[27] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[28] Assaf Zomet, Doron Feldman, Shmuel Peleg, and Daphna Weinshall. Mosaicing new views: The crossed-slits projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:741–754, 2003.

Part II

Publications

Chapter A

Rectifying rolling shutter video from hand-held devices

This is an edited version of the paper:

> Per-Erik Forssén and Erik Ringaby. Rectifying rolling shutter video from hand-held devices. In *IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco, USA, 2010. IEEE Computer Society.

Rectifying rolling shutter video from hand-held devices

Per-Erik Forssén and Erik Ringaby
Department of Electrical Engineering, Computer Vision Laboratory
Linköping University, Sweden

Abstract

This paper presents a method for rectifying video sequences from *rolling shutter* (RS) cameras. In contrast to previous RS rectification attempts we model distortions as being caused by the 3D motion of the camera. The camera motion is parametrised as a continuous curve, with knots at the last row of each frame. Curve parameters are solved for using non-linear least squares over inter-frame correspondences obtained from a KLT tracker. We have generated synthetic RS sequences with associated ground-truth to allow controlled evaluation. Using these sequences, we demonstrate that our algorithm improves over to two previously published methods. The RS dataset is available on the web to allow comparison with other methods.

1 Introduction

Today consumer products that allow video capture are quite common. Examples are cell-phones, music players, and regular cameras. Most of these devices, as well as camcorders in the consumer price range, have CMOS image sensors. CMOS sensors have several advantages over the conventional CCD sensors: they are cheaper to manufacture, and typically offer on-chip processing [9], for e.g. automated white balance and auto-focus measurements. However, most CMOS sensors, by design make use of what is known as a *rolling shutter* (RS). In an RS camera, detector rows are read and reset sequentially. As the detectors collect light right until the time of readout, this means that each row is exposed during a slightly different time window. The more conventional CCD sensors on the other hand use a *global shutter* (GS), where all pixels are reset simultaneously, and collect light during the same time interval. The downside with a rolling shutter is that since pixels are acquired at different points in time, motion of either camera or target will cause geometrical distortions in the acquired images. Figure 1 shows an example of geometric distortions caused by using a rolling shutter, and how this frame is rectified by our proposed method, as well as two others.

Figure 1: Example of rolling shutter imagery. Top left: Frame from an iPhone 3GS camera sequence acquired during fast motion. Top right: Rectification using our rotation method. Bottom left: Rectification using the global affine method. Bottom right: Rectification using the global shift method. Videos are available on the web and in the supplementary material.

1.1 Related work

A camera motion between two points in time can be described with a three element translation vector, and a 3DOF (degrees-of-freedom) rotation. For hand-held footage, the rotation component is typically the dominant cause of image plane motion. (A notable exception to this is footage from a moving platform, such as a car.) Many new camcorders thus have *mechanical image stabilisation* (MIS) systems that move the lenses (some instead move the sensor) to compensate for small pan and tilt rotational motions (image plane rotations, and large motions, are not handled). The MIS parameters are typically optimised to the frequency range caused by a person holding a camera, and thus work well for such situations. However, since lenses have a certain mass, and thus inertia, MIS has problems keeping up with faster motions, such as caused by vibrations from a car engine. Furthermore, cell phones, and lower end camcorders lack MIS. There is thus a large volume of video out there, that exhibit RS artifacts.

For cases when MIS is absent, or non-effective, one can instead do post-capture image rectification. There exist a number of different approaches for dealing with

special cases of this problem [5, 13, 16, 6, 7, 8]. Some algorithms assume that the image deformation is caused by a globally constant translational motion across the image [5, 16, 8]. After rectification this would correspond to a constant optical flow across the entire image, which is rare in practise. Liang et al. [13] improve on this by giving each row a different motion, that is found by interpolating between constant global inter-frame motions using a Bézier curve. Another improvement is due to Cho et al. [6, 7]. Here geometric distortion is modelled as a global affine deformation that is parametrised by the scan-line index.

All current RS rectification approaches perform warping of individual frames to rectify RS imagery. Note that a perfect compensation under camera translation would require the use of multiple images, as the parallax induced by a moving camera will cause occlusions. Single frame approximations do however have the advantage that ghosting artifacts caused by multi-frame reconstruction is avoided, and is thus preferred in the related problem of video stabilisation [14].

Other related work on RS images include structure and motion estimation. Geyer et al. [10] study the projective geometry of RS cameras, and also describe a calibration technique for estimation of the readout parameters. The derived equations are then used for structure and motion estimation in synthetic RS imagery. Ait-Aider et al. demonstrate that motion estimation is possible from single rolling shutter frames if world point-to-point distances are known, or from curves that are known to be straight lines in the world [1]. They also demonstrate that structure and motion estimation can be done from a single stereo pair if one of the used cameras has a rolling shutter [2].

1.2 Contributions

All the previous approaches to rectification of RS video [5, 13, 16, 6, 7, 8] model distortions as taking place in the image plane. We instead model the 3D camera motion using calibrated projective geometry. We introduce two models, one purely rotational, and one with rotation and translation with respect to an estimated plane in the scene. In projective geometry terms, these can be thought of as a sequence of parametrised homographies, one for each image row.

This far, no controlled comparison of RS algorithms have been published. Instead each new algorithm has just been published together with images that show how well images distorted by particular motions can be rectified. In related fields such as stereo, and optical flow computation [3], evaluation datasets have been important for ensuring that new algorithms actually improve on previous ones. For these reasons we have generated synthetic RS sequences, with associated ground-truth rectifications. Using these sequences, we compare our own implementations of the global affine model [6], and the global shift model [8] to the new method that we propose. Our dataset and supplementary videos are available for download at [18].

1.3 Overview

This paper is organised as follows: In section 2, we describe how to calibrate a rolling-shutter camera, and introduce models and cost functions for camera ego-motion estimation. In section 3 we discuss interpolation schemes for rectification of rolling-shutter

imagery. In section 4 we describe our evaluation dataset. In section 5 we use our dataset to compare different interpolation schemes, and to compare our camera ego-motion approach to our own implementations of [13] and [8]. The paper concludes with outlooks and concluding remarks in section 6.

2 Camera motion estimation

In this paper, we take the *intrinsic camera matrix*, the *camera frame-rate* and the *inter-frame delay* to be given. This reduces the number of parameters that need to be estimated on-line, but also requires us to calibrate each camera before the algorithms can be used.

2.1 Rolling shutter camera calibration

A 3D point, $\mathbf{X}$, and its projection in the image, $\mathbf{x}$, given in homogeneous coordinates, are related according to

$$\mathbf{x} = \mathbf{K}\mathbf{X} \ , \text{ and } \quad \mathbf{X} = \lambda \mathbf{K}^{-1}\mathbf{x} \ , \tag{1}$$

where $\mathbf{K}$ is a 5DOF upper triangular 3×3 *intrinsic camera matrix*, and λ is an unknown scaling [12]. We estimate $\mathbf{K}$ using the Zhang method in OpenCV. [21].

The RS chip frame period $1/f$ (where f is the *frame rate*) is divided into a *readout time* t_r, and an *inter-frame delay*, t_d as: $1/f = t_r + t_d$. The readout time can be calibrated by imaging a flashing light source with known frequency [10], see figure 2, left. If we measure the period T of the vertical oscillation in pixels, t_r can be obtained as:

$$t_r = N_r/(T f_o) \, , \tag{2}$$

where N_r is the number of image rows, and f_o is the oscillator frequency. The inter-frame delay can now be computed as $t_d = 1/f - t_r$. For our purposes it is preferable to use rows as fundamental unit, and express the inter-frame delay as a number of *blank rows*:

$$N_b = N_r t_d/(1/f) = N_r(1 - t_r f) \, . \tag{3}$$

We have performed both Zhang [21], and Geyer [10] calibrations for the cameras built into the Apple iPhone 3GS, and the SonyEricsson W890i cell phones. As the Geyer calibration is a bit awkward (it requires a signal generator, an oscilloscope and an LED), we have reproduced the calibration values we obtained in table 1. The camera frame rates are $f = 30$ Hz for the 3GS, and $f = 14.7059$ Hz for the W890i (according to manufacturer specifications).

In his paper [10], Geyer suggests removing the lens, in order to get a homogeneous illumination of the sensor. This is difficult to do on cellphones, and thus we instead recommend to collect a sequence of images of the flashing LED, and then subtract the average image from each of these, see figure 2, right. This removes most of the shading seen in figure 2, left, and allows us to find the oscillation period from the first frequency above DC.

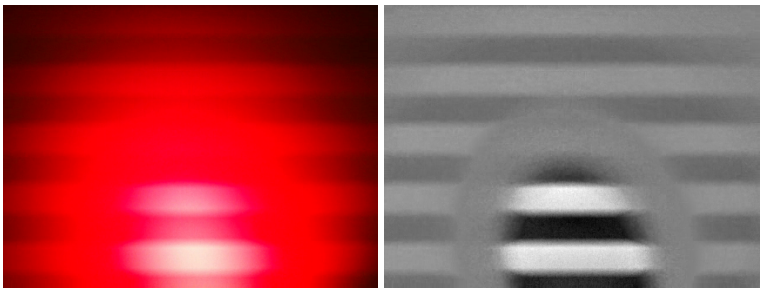


Figure 2: Calibration of a rolling-shutter camera. Left: Image of a flashing LED used for calibration of the readout time. Right: Corresponding image after subtraction of the temporal average.

Camera	f_o	t_r
W890i	5.02	60.94
	6.5	60.76
	7.5	60.88
	avg.	60.86

Camera	f_o	t_r
3GS	4.01	30.54
	5.01	31.22
	6.5	30.5
	avg.	30.75

Table 1: Used oscillator frequencies f_o, and obtained readout times t_r, for the SonyEricsson W890i and Apple iPhone 3GS camera phones. Units are milliseconds.

2.2 Rolling shutter camera under pure rotation

Our first model of camera motion is a rotation about the camera centre during frame capture, in a smooth, but time varying way. We represent this as a sequence of rotation matrices, $\mathbf{R}(t) \in \mathrm{SO}(3)$.

Two homogeneous image points $\mathbf{x}$, and $\mathbf{y}$, that correspond in consecutive frames, are now expressed as:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(t_1)\mathbf{X}\ ,\ \text{and}\quad \mathbf{y} = \mathbf{K}\mathbf{R}(t_2)\mathbf{X}\,. \tag{4}$$

This gives us the relation:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(t_1)\mathbf{R}^T(t_2)\mathbf{K}^{-1}\mathbf{y}\,. \tag{5}$$

The time parameter is a linear function of the current image row (i.e. x_2/x_3 and y_2/y_3). Thus, by choosing the unit of time as image rows, and time zero as the top row of the first frame, we get $t_1 = x_2/x_3$. In the second image we get $t_2 = y_2/y_3 + N_r + N_b$, where N_r is the number of image rows, and N_b is defined in (3).

Each correspondence between the two views, (5) gives us two equations (after elimination of the unknown scale) where the unknowns are the rotations. Unless we constrain the rotations further, we now have six unknowns (a rotation can be parametrised with three parameters) for each correspondence. We thus parametrise the rotations with an interpolating spline with knots at the last row of each frame, see figure 3. Intermediate rotations are found using spherical linear interpolation [20].

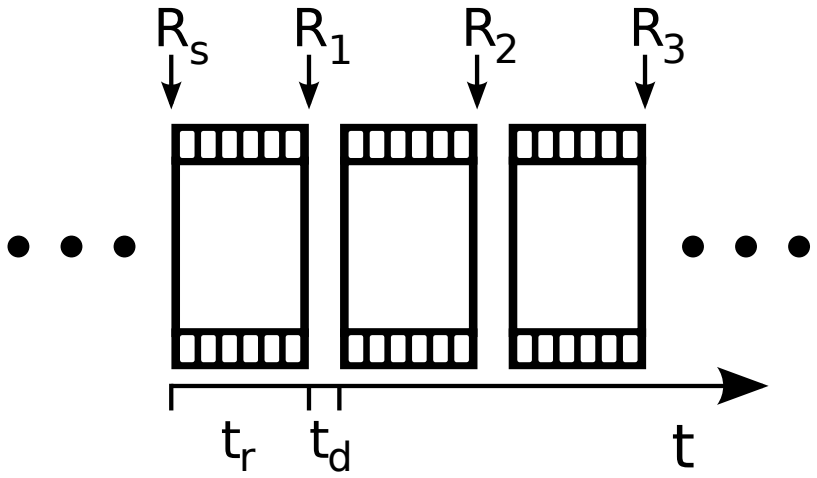


Figure 3: Rotations, $\mathbf{R}_1, \mathbf{R}_2, \ldots$ are estimated for last rows of each frame. Intermediate rotations are interpolated from these and $\mathbf{R}_s$. Readout time t_r, and inter-frame delay t_d are also shown.

As we need a reference world frame, we might as well fixate that to the start of frame 1, i.e. set $\mathbf{R}_s = \mathbf{I}$. This gives us $3N$ unknowns in total for a group of N frames. These can be resolved if we have at least three correspondences between each pair of views.

2.3 Rolling shutter imaging of a planar scene

In our second camera motion model, we assume that we are imaging a purely planar scene. We now model the motion as a sequence of translations $\mathbf{d}(t) \in \mathbb{R}^3$, and rotations $\mathbf{R}(t) \in \mathrm{SO}(3)$, with respect to a coordinate system located on the world plane. The world coordinate system needs not be explicitly estimated, it suffices to know that we can choose it such that the 3D points have a zero third coordinate, i.e. $(0\ 0\ 1)\,\mathbf{X} = 0$. The projection of such a point in the image, after a translation $\mathbf{d}(t_1)$, and a rotation $\mathbf{R}(t_1)$, can be written:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(t_1)(\mathbf{X} + \mathbf{d}(t_1)) = \mathbf{K}\mathbf{R}(t_1)\mathbf{D}(t_1)\tilde{\mathbf{X}}\,, \tag{6}$$

$$\text{where} \quad \mathbf{D} = \begin{pmatrix} 1 & 0 & d_1 \\ 0 & 1 & d_2 \\ 0 & 0 & d_3 \end{pmatrix}\,, \tag{7}$$

and $\tilde{\mathbf{X}}$ is a three element vector containing the non-zero elements of $\mathbf{X}$, and a 1 in the third position.

Since (6) is invertible, in the same sense as (1), we can relate the projections of the

3D point in two images as:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(t_1)\mathbf{D}(t_1)\mathbf{D}(t_2)^{-1}\mathbf{R}(t_2)^T\mathbf{K}^{-1}\mathbf{y}\,. \tag{8}$$

Note that by setting $\mathbf{D}(t_1) = \mathbf{D}(t_2) = \mathbf{I}$ we obtain the pure rotation model (5) as a special case.

In contrast to the pure rotation case, we have now a variable origin, so we need also to find $\mathbf{R}_s$ and $\mathbf{d}_s$. However, we note that a point in the world plane, expressed in normalised camera coordinates $\mathbf{X}_c = \lambda\mathbf{K}^{-1}\mathbf{x}$, has to satisfy a plane equation:

$$\hat{\mathbf{r}}^T\mathbf{X}_c - \rho = \hat{\mathbf{r}}^T(\mathbf{X}_c - \hat{\mathbf{r}}\rho) = 0\,. \tag{9}$$

We now let this equation define the transformation from the camera to the third (zero valued) world coordinate:

$$\mathbf{X} = \mathbf{R}_s^T(\mathbf{X}_c - \hat{\mathbf{r}}\rho) \;\text{ for }\; \mathbf{R}_s = \begin{pmatrix}\hat{\mathbf{r}}_\perp & \hat{\mathbf{r}}\times\hat{\mathbf{r}}_\perp & \hat{\mathbf{r}}\end{pmatrix}\,. \tag{10}$$

This gives us the projection from the plane into the camera as:

$$\mathbf{X}_c = \mathbf{R}_s(\mathbf{X} + (0\;0\;\rho)^T)\,. \tag{11}$$

Finally, as a monocular reconstruction is only defined up to scale, we can fixate the plane at $\rho = 1$. This locks the translation to $\mathbf{d}_s = (0\;0\;1)^T$, and we thus only get the extra 3 parameters in $\mathbf{R}_s$.

Just like in the pure rotation case, each correspondence gives us two equations, but now we have $6N + 3$ unknowns for a group of N frames. These can be determined if we have at least $3N + 2$ correspondences, and at least 6 correspondences between each pair of frames (8 is required for the first pair).

2.4 Pure translational motion

It is also possible to constrain the planar scene model to translations only. For this we simply set all rotation matrices equal to the first, i.e $\mathbf{R}_n = \mathbf{R}_s\ \forall n \in [1, N]$. This gives us $3N + 3$ unknowns, which again requires at least 3 correspondences between each pair of frames.

2.5 Motion interpolation

We interpolate the translational component of the camera motion, $\mathbf{d}(t) \in \mathbb{R}^3$ in-between two key translations $\mathbf{d}_1$, $\mathbf{d}_2$, using regular linear interpolation. Using a parameter $w \in [0, 1]$, this can be written as:

$$\mathbf{d}_{\text{interp}} = (1 - w)\mathbf{d}_1 + w\mathbf{d}_2\,. \tag{12}$$

For the rotational component, the situation is more complicated, due to the periodic structure of SO(3).

We have chosen to represent rotations as three element vectors where the magnitude corresponds to the rotation angle, and the direction is the axis of rotation, i.e. $\mathbf{n} = \phi\hat{\mathbf{n}}$.

This is a minimal parametrisation of rotations, and it also ensures smooth variations, in contrast to e.g. Euler angles. It is thus suitable for parameter optimisation. The vector $\mathbf{n}$ can be converted to a rotation matrix using the matrix exponent, which for a rotation simplifies to Rodrigues formula:

$$\mathbf{R} = \text{expm}(\mathbf{n}) = \mathbf{I} + [\hat{\mathbf{n}}]_x \sin\phi + [\hat{\mathbf{n}}]_x^2(1-\cos\phi) \tag{13}$$

$$\text{where } [\hat{\mathbf{n}}]_x = \frac{1}{\phi}\begin{pmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{pmatrix} . \tag{14}$$

Conversion back to vector form is accomplished through the matrix logarithm in the general case, but for a rotation matrix, there is a closed form solution. We note that two of the terms in (13) are symmetric, and thus terms of the form $r_{ij} - r_{ji}$ will come from the anti-symmetric part alone. This allows us to extract the axis and angle as:

$$\mathbf{n} = \text{logm}(\mathbf{R}) = \phi\hat{\mathbf{n}}\,, \quad \text{where} \quad \begin{cases} \tilde{\mathbf{n}} = \begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix} \\ \phi = \sin^{-1}(||\tilde{\mathbf{n}}||/2) \\ \hat{\mathbf{n}} = \tilde{\mathbf{n}}/||\tilde{\mathbf{n}}||\,. \end{cases} \tag{15}$$

It is also possible to extract the rotation angle from the trace of $\mathbf{R}$ [17]. We recommend (15), as it avoids numerical problems for small angles. Using (13) and (15), we can perform SLERP (Spherical Linear intERPolation) [20] between two rotations $\mathbf{n}_1$ and $\mathbf{n}_2$, using an interpolation parameter $w \in [0,1]$ as follows:

$$\mathbf{n}_{\text{diff}} = \text{logm}\left(\text{expm}(-\mathbf{n}_1)\text{expm}(\mathbf{n}_2)\right) \tag{16}$$

$$\mathbf{R}_{\text{interp}} = \text{expm}(\mathbf{n}_1)\text{expm}(w\mathbf{n}_{\text{diff}}) \tag{17}$$

2.6 Optimisation

We now wish to solve for the unknown motion parameters, using iterative minimisation. For this we need a cost function:

$$J = \epsilon(\mathbf{n}_1, \ldots, \mathbf{n}_N) \;\text{ or} \tag{18}$$

$$J = \epsilon(\mathbf{n}_s, \mathbf{n}_1, \ldots, \mathbf{n}_N, \mathbf{d}_1, \ldots, \mathbf{d}_N)\,, \tag{19}$$

for the pure rotation, and the planar scene models respectively. To this end, we choose to minimise the (symmetric) image-plane residuals of the set of corresponding points $\mathbf{x}_k \leftrightarrow \mathbf{y}_k$:

$$J = \sum_{k=1}^{K} d(\mathbf{x}_k, \mathbf{H}\mathbf{y}_k)^2 + d(\mathbf{y}_k, \mathbf{H}^{-1}\mathbf{x}_k)^2 \tag{20}$$

$$\text{where } \mathbf{H} = \mathbf{K}\mathbf{R}(\mathbf{x}_k)\mathbf{R}^T(\mathbf{y}_k)\mathbf{K}^{-1} \;\text{ or} \tag{21}$$

$$\mathbf{H} = \mathbf{K}\mathbf{R}(\mathbf{x}_k)\mathbf{D}(\mathbf{x}_k)\mathbf{D}(\mathbf{y}_k)^{-1}\mathbf{R}^T(\mathbf{y}_k)\mathbf{K}^{-1} \tag{22}$$

Here the distance function $d(\mathbf{x}, \mathbf{y})$ for homogeneous vectors, is given by:

$$d(\mathbf{x}, \mathbf{y})^2 = (x_1/x_3 - y_1/y_3)^2 + (x_2/x_3 - y_2/y_3)^2 \,. \tag{23}$$

The rotation matrices are obtained as:

$$\mathbf{R}(\mathbf{x}) = \texttt{SLERP}(\mathbf{n}_1, \mathbf{n}_2, w) \,, \text{ for } \; w = x_2/(x_3 N_r), \tag{24}$$

where `SLERP` is defined in (16,17), and N_r is the number of image rows.

In our experiments, we have minimised (20) using the MATLAB optimiser function `lsqnonlin`. Rewriting the optimisation in e.g. C should however be done if real-time operation is to be achieved.

For speed, we have chosen to optimise over short intervals of $N = 2$, 3 or 4 frames. For the pure rotation model, there is a simple way to initialise a new interval from the previous one. Once the optimiser has found a solution for a group of frames, we change the origin to the second camera in the sequence (see figure 3), i.e.

$$\mathbf{R}_o = \texttt{SLERP}(\mathbf{n}_1, \mathbf{n}_2, N_b/(N_r + N_b)) \,. \tag{25}$$

Then we shift the interval one step, correct for the change of origin, and use the previous rotations as initialisations

$$\mathbf{R}'_n = \mathbf{R}_o^T \mathbf{R}_{n+1} \,, \quad \text{for } \; n = 1, \ldots, N \,. \tag{26}$$

As initialisation of the rotations in newly shifted-in frames, we use identity rotations.

In the planar scene model, we initialise the rotations to identity rotations, and the translations to $\mathbf{d}_n = (0 \; 0 \; 1)^T \;\; \forall n \in [1, N]$.

2.7 Point correspondences

The point correspondences needed to estimate the rotations are obtained through point tracking. First, Harris-points [11] are detected in the current frame and these are tracked using the KLT tracker [15, 19]. The KLT tracker uses a spatial intensity gradient search which minimises the Euclidean distance between the corresponding patches in the consecutive frames. We use the scale pyramid implementation of the algorithm in OpenCV. Using pyramids makes it easier to detect large movements.

To increase the accuracy of the point tracker, a track-re-track procedure is used [3]. When the points have been tracked from the first image to the other, the tracking is reversed and only the points that return to the original position (within a threshold) are kept. The computation cost is doubled but outliers are removed effectively.

3 Image rectification

Once we have found our sequence of rotation matrices, we can use them to rectify the images in the sequence. Each row gets its own rotation according to (24). We can then align them to a reference row $\mathbf{R}_o$ (typically the middle row), using:

$$\mathbf{R}'(\mathbf{x}) = \mathbf{R}_o \mathbf{R}^T(\mathbf{x}) \,. \tag{27}$$

This gives us the forward mapping as:

$$\mathbf{x}' = \mathbf{K}\mathbf{R}_o\mathbf{R}^T(\mathbf{x})\mathbf{K}^{-1}\mathbf{x} \tag{28}$$

This tells us how each point should be displaced in order to rectify the scene. Using this relation we can transform all the pixels to their new, rectified locations.

We have chosen to perform the rectifying interpolation in three steps: First, we create an all-zero RGBA image. Second, we apply (28) to each pixel in the RS image. The 3×3 closest grid locations are then updated by adding vectors of the form (wr, wg, wb, w). Here r, g, b are the colour channel values of the input pixel, and w is a variable weight that depends on the grid location $\mathbf{u}$, according to:

$$w(\mathbf{u}) = \exp(-.5||\mathbf{u} - \tilde{\mathbf{x}}'||^2/\sigma^2)\,. \tag{29}$$

Here $\tilde{\mathbf{x}}' = (x'_1/x'_3 \;\; x'_2/x'_3)^T$ is the sub-pixel location of the pixel, and σ is a smoothing parameter, which we set to $\sigma = 0.15$. Third, after looping through all pixels, we convert the RGBA image to RGB, by dividing the RGB values by the fourth element. This *forward interpolation* scheme is quite fast, and its parallel nature makes it well suited to a GPU implementation.

Alternatively, the irregular grid of pixels can be resampled to a regular grid, by defining a triangular mesh over the points, and sampling the mesh using bicubic interpolation. This is done by the function `griddata` in Matlab.

Finally, it is also tempting to use regular, or *inverse interpolation*, i.e. invert (28) to obtain:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(\mathbf{x}')\mathbf{R}_o^T\mathbf{K}^{-1}\mathbf{x}'\,. \tag{30}$$

We can now loop over all values of $\mathbf{x}'$, and use (30) to find the pixel locations in the distorted image, and cubically interpolate these.

4 Synthetic dataset

In order to do a controlled evaluation of algorithms for RS compensation we have generated six test sequences (available at [18]), using the Autodesk Maya software package. Each sequence consists of 12 RS distorted frames of size 640×480, corresponding ground-truth *global shutter* (GS) frames, and masks that indicate pixels in the ground-truth frames that can be reconstructed from the corresponding rolling-shutter frame. In order to suit all algorithms, the ground-truth frames and masks come in three variants: for rectification to the time instant when the first, middle and last row of the RS frame were imaged.

Each synthetic RS frame is created from a GS sequence with one frame for each RS image row. One row in each GS image is used, starting at the top row and sequentially down to the bottom row. In order to simulate an inter-frame delay, we also generate a number of GS frames that are not used to build any of the RS frames. The camera is however still moving during these frames.

We have generated four kinds of synthetic sequences, using different camera motions in a static scene, see figure 4.

Figure 4: The four categories of synthetic sequences.

In the first sequence type, the camera rotates around its centre in a spiral fashion, see figure 4 top left. Three different versions of this sequence exist to test the importance of modelling the inter-frame delay. The different inter-frame delays are $N_b = 0$, 20 and 40 *blank rows* (i.e. the number of unused GS frames).

In the second sequence type, the camera makes a pure translation to the right and has an inter-frame delay of 40 blank rows, see figure 4 top right.

In the last two sequence types the camera makes an up/down rotating movement, with a superimposed rotation from left to right, see figure 4 bottom left. There is also a back-and-forth rotation with the viewing direction as axis. The last sequence type is the same as the third except that a translation parallel to the image plane has been added, see figure 4 bottom right.

For each frame in the ground-truth sequences, we have created masks that indicate pixels that can be reconstructed from the corresponding RS frame, see figure 5. These masks were rendered by inserting one light source for each image row, into an otherwise dark scene. The light sources had a rectangular shape that illuminates exactly the part of the scene that was imaged by the RS camera when located at that particular place. To acquire the mask, a global shutter render is triggered at the desired location (e.g. corresponding to first, middle or last row in the RS-frame).

Figure 5: Left to right: Rendered RS frame from sequence of type #2, with $N_b = 40$ (note that everything is slightly slanted to the right), corresponding global-shutter ground-truth, and mask with white for ground-truth pixels that were seen in the RS frame.

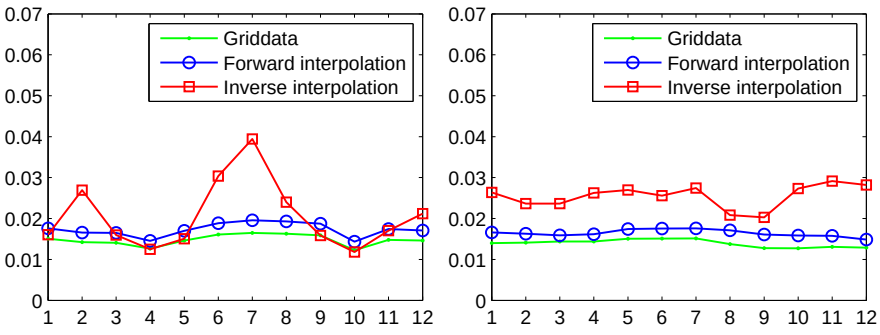


Figure 6: Comparison of interpolation scheme errors. The plots show the average Euclidean pixel distance between interpolated images and rendered ground truth for each frame in sequence type #1, $N_b = 0$ (left), and type #3, $N_b = 40$ (right).

5 Experiments

5.1 Interpolation accuracy

We have compared the errors of the three interpolation approaches described in section 3, in figure 6. Here we have used known ground-truth rotations to rectify each frame in two pure camera rotation sequences, sequence type #1, with $N_b = 0$, and sequence type #3, with $N_b = 40$ (see section 4 for a description of the sequences). We have used two pure rotation sequences, as for these an almost perfect reconstruction is possible, and thus the errors shown are due to interpolation only. The error measure used is average Euclidean distance to the RGB pixel values in the ground truth images, within the valid mask.

In some frames, the methods differ quite a bit, while in others they are very similar. The reason for this is that only for larger rotations, do the neighbours in the distorted and undistorted images start to differ. As can be seen in figure 6, `griddata` and our forward interpolation are superior to inverse sampling. Among the three methods, `griddata` stands out, by being approximately $40\times$ more expensive on 640×480 images. As our forward interpolation scheme is both fast and accurate, we recommend it over the other methods.

For very fast motions, and a slow rolling shutter, the 3×3 grid used in forward interpolation may be too small. The interpolated image would then have pixels where the value is undefined. In our experiments on real video we have however not experienced this. Should this effect occur, one could simply increase the grid size to 5×5.

5.2 Rectification accuracy

We have compared our methods to the *global affine model* (GA) [6], and the *global shift model* (GS) [8] on our synthetic sequences, see section 4. The comparison is done using thresholded Euclidean colour distance. Pixels that deviate more than $d_{\text{thr}} = 0.3$ are counted as incorrect. We have also tried other threshold values, and while the exact choice changes the locations of the curves, it does not change their order (for reasonable values of d_{thr}). As evaluation measure we use the fraction of correctly reconstructed pixels within the mask of valid locations. For clarity of presentation, we only present a subset of the results on our synthetic dataset. As a baseline, all plots contain the errors for uncorrected frames, with respect to the first frame ground-truth.

As our reconstruction solves for several cameras in each frame interval, we have simply chosen to present all of them in the following plots. E.g. Rotation 1, 2, and 3 in figure 7 are the three solutions in a 3-frame reconstruction.

In figure 7 we compare the GA, and GS methods with our pure rotation model. The sequence used is type #1 (rotation only), with $N_b = 0$. As can be seen, our methods do better than GA, GS, and the baseline.

In figure 8 we compare the GA, and GS methods with our pure rotation model on sequence type #3 (rotation only), with $N_b = 40$. As can be seen our methods do better than both GA, GS, and the baseline. GA, and GS, have problems with this sequence, and sometimes fall below the baseline. In general, other values of N_b give very similar

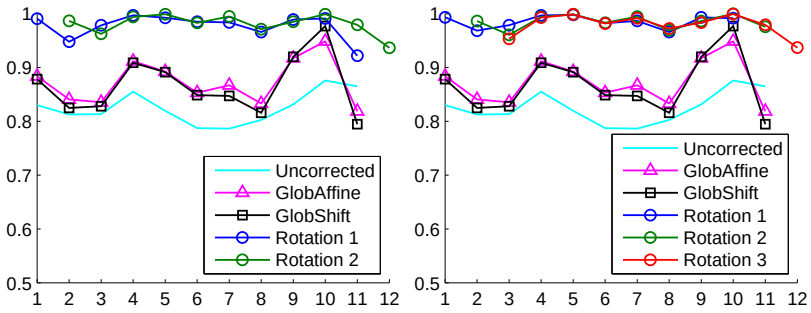


Figure 7: Sequence type #1 (rotation only), $N_b = 0$. Left: GA, GS, and uncorrected frames, against our **rotation model** with 2-frame reconstruction window. Right: GA, GS, and uncorrected frames, against our **rotation model** with 3-frame window.

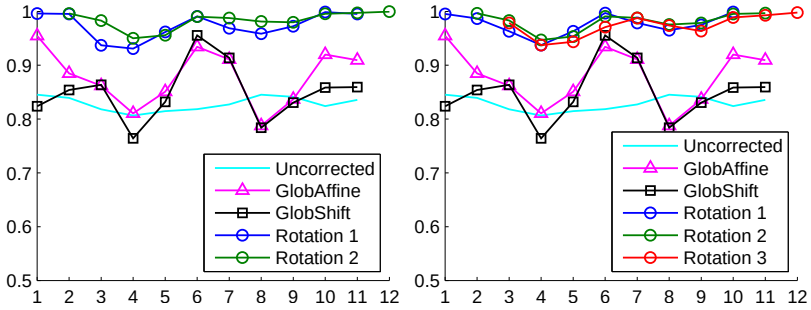


Figure 8: Sequence type #3 (rotation only), $N_b = 40$. Left: GA, GS, and uncorrected frames, against our **rotation model** with 2-frame reconstruction window. Right: GA, GS, and uncorrected frames, against our **rotation model** with 3-frame window.

results for our methods. For GA and GS the variability is larger, but we have not seen any consistent degradation or improvement.

In figure 9, left, we compare the GA, and GS methods with our pure rotation model. The sequence used is type #2 (translation only), with $N_b = 40$. As can be seen our methods do slightly worse than GA and GS, but they still improve on the uncorrected input. In figure 9, right, we compare GA, GS, and our translation-only model. The translation reconstruction for the first frame is still worse than GA and GS, but the other two do significantly better.

In figure 10 we have compared GA, GT, with our rotation only model (left) and with the full model (right). As can be seen, the rotation only model does consistently better than the others. Note that the full model currently does worse than the rotation only model. When we gave the optimiser different starting points, (e.g. the result from the rotation model) we obtained different solutions, thus we conclude that the cost function for the full model is not convex. A better initialisation may solve this problem, but this is out of the scope of this paper.

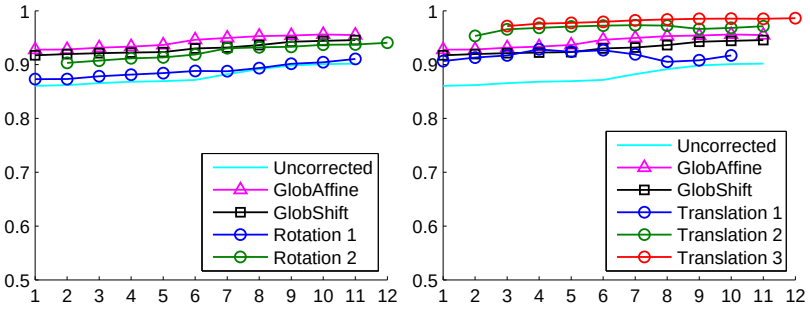


Figure 9: Sequence type #2 (translation only), $N_b = 40$. Left: GA, GS, and uncorrected frames, against our **rotation model** with 2-frame reconstruction window. Right: GA, GS, and uncorrected frames, against our **translation model** with 3-frame window.

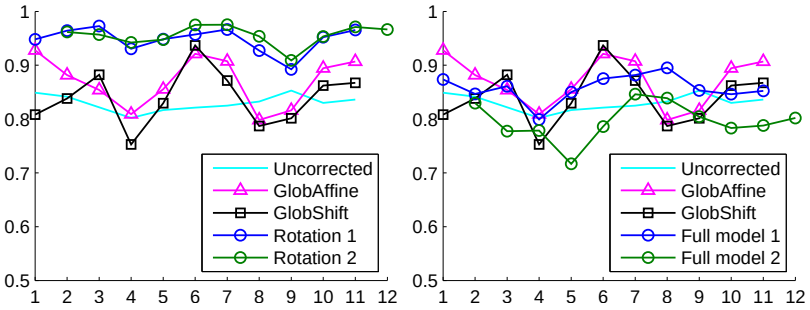


Figure 10: Sequence type #4 (translation and rotation), $N_b = 40$. Left: GA, GS, and uncorrected frames, against our **rotation model** with 2-frame reconstruction window. Right: GA, GS, and uncorrected frames, against our **full model** with 2-frame window.

Figure 11: Image stabilisation. Left: Uncorrected RS frame. Centre: Rectified frame, with tracked points indicated. Right: Frame stabilised by centring the tracked points along a vertical line.

5.3 Stabilisation of rolling-shutter video

We have done a simple comparison of RS compensation algorithms on real imagery, using image stabilisation. Such a comparison requires that the imaged scene is static, and that the camera translation is negligible. We do this by tracking two points through the RS frames, using the KLT-tracker [15, 19]. After rolling-shutter compensation, we perform a virtual rotation of the frames (using a global homography), such that two points in the scene are placed symmetrically about the image centre, along a vertical line, see figure 11. The only manual input to this approach is that the two points are indicated manually in the first frame.

We supply two such stabilised sequences as supplemental material (one for the iPhone 3GS and one from the W890i), together with the corresponding uncorrected RS sequences, and results for the GA and GS methods. A single frame comparison of the rectification step is also shown in figure 1, for the iPhone 3GS.

6 Concluding remarks

In this paper, we have demonstrated rolling-shutter rectification by modelling the camera motion, and shown this to be superior to techniques that model movements in the image plane only. We even saw that image-plane techniques occasionally perform worse than the uncorrected baseline. This is especially true for motions that they do not model, e.g. rotations for the Global shift model [8].

The method we currently see as the best one is the rotation only model. In addition to being the overall best method, it is also the fastest of our models. Note that even this model corrects for more types of camera motion than does mechanical image stabilisation (MIS). In future work we plan to improve our approach by replacing the linear interpolation with a higher order spline. We will also investigate better initialisations for the full model. Another obvious improvement is to optimise parameters over full sequences. However, we wish to stress that our aim is currently to allow the algorithm to run on mobile platforms, which excludes optimisation over longer frame intervals than the 2-4 that we currently use.

In general, the quality of the reconstruction should benefit from more measurements. In MIS systems, camera rotations are measured by MEMS gyro sensors [4].

It would be interesting to see how such measurements could be combined with measurements from KLT-tracking when rectifying video. There are also accelerometers in many cellphones, and measurements from these could also be useful in ego-motion estimation.

References

[1] Omar Ait-Aider, Adrien Bartoli, and Nicolas Andreff. Kinematics from lines in a single rolling shutter image. In *CVPR'07*, Minneapolis, USA, June 2007.

[2] Omar Ait-Aider and Francois Berry. Structure and kinematics triangulation with a rolling shutter stereo rig. In *IEEE International Conference on Computer Vision*, 2009.

[3] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. In *IEEE ICCV*, 2007.

[4] Jonathan Bernstein. An overview of MEMS inertial sensing technology. *Sensors Magazine*, 2003(1), February 2003.

[5] Li-Wen Chang, Chia-Kai Liang, and H.H. Chen. Analysis and compensation of rolling shutter distortion for CMOS image sensor arrays. In *ISCOM'05*, 2005.

[6] Won-Ho Cho and Ki-Sang Hong. Affine motion based CMOS distortion analysis and CMOS digital image stabilization. *IEEE TCE*, 53(3):833–841, August 2007.

[7] Won-Ho Cho, Dae-Woong Kim, and Ki-Sang Hong. CMOS digital image stabilization. *IEEE TCE*, 53(3):979–986, 2007.

[8] Jung-Bum Chun, Hunjoon Jung, and Chong-Min Kyung. Suppressing rolling-shutter distortion of CMOS image sensors by motion vector detection. *IEEE TCE*, 54(4):1479–1487, 2008.

[9] Abbas El Gamal and Helmy Eltoukhy. CMOS image sensors. *IEEE Circuits and Devices Magazine*, May/June 2005.

[10] Christopher Geyer, Marci Meingast, and Shankar Sastry. Geometric models of rolling-shutter cameras. In *6th OmniVis WS*, 2005.

[11] C. G. Harris and M. Stephens. A combined corner and edge detector. In *4th Alvey Vision Conference*, pages 147–151, September 1988.

[12] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[13] Chia-Kai Liang, Li-Wen Chang, and H.H. Chen. Analysis and compensation of rolling shutter effect. *IEEE Transactions on Image Processing*, 17(8):1323–1330, August 2008.

[14] Feng Liu, Michael Gleicher, Hailin Jin, and Aseem Agarwala. Content-preserving warps for 3D video stabilization. In *ACM Transactions on Graphics*, 2009.

[15] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI81*, pages 674–679, 1981.

[16] Steven P. Nicklin, Robin D. Fisher, and Richard H. Middleton. Rolling shutter image compensation. In *Robocup 2006 LNAI 4434*, pages 402–409, 2007.

[17] F.C. Park and Bahram Ravani. Smooth invariant interpolation of rotations. *ACM Transactions on Graphics*, 16(3):277–295, July 1997.

[18] Erik Ringaby. Rolling shutter dataset with ground truth. http://www.cvl.isy.liu.se/research/rs-dataset.

[19] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR'94*, Seattle, June 1994.

[20] Ken Shoemake. Animating rotation with quaternion curves. In *Int. Conf. on CGIT*, pages 245–254, 1985.

[21] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

Chapter B

Efficient Video Rectification and Stabilisation for Cell-Phones

This is an edited version of the paper:

Erik Ringaby and Per-Erik Forssén. Efficient video rectification and stabilisation for cell-phones. *International Journal of Computer Vision*, 96(3):335–352, 2012.

The final publication is available at springerlink.com
Digital Object Identifier: 10.1007/s11263-011-0465-8

Efficient Video Rectification and Stabilisation for Cell-Phones

Erik Ringaby and Per-Erik Forssén
Department of Electrical Engineering, Computer Vision Laboratory
Linköping University, Sweden

Abstract

This article presents a method for rectifying and stabilising video from cell-phones with *rolling shutter* (RS) cameras. Due to size constraints, cell-phone cameras have constant, or near constant focal length, making them an ideal application for calibrated projective geometry. In contrast to previous RS rectification attempts that model distortions in the image plane, we model the 3D rotation of the camera. We parameterise the camera rotation as a continuous curve, with knots distributed across a short frame interval. Curve parameters are found using non-linear least squares over inter-frame correspondences from a KLT tracker. By smoothing a sequence of reference rotations from the estimated curve, we can at a small extra cost, obtain a high-quality image stabilisation. Using synthetic RS sequences with associated ground-truth, we demonstrate that our rectification improves over two other methods. We also compare our video stabilisation with the methods in iMovie and Deshaker.

1 Introduction

Almost all new cell-phones have cameras with CMOS image sensors. CMOS sensors have several advantages over the conventional CCD sensors: notably they are cheaper to manufacture, and typically offer on-chip processing [13], for e.g. automated white balance and auto-focus measurements.

However, most CMOS sensors, by design make use of what is known as a *rolling shutter* (RS). In an RS camera, detector rows are read and reset sequentially. As the detectors collect light right until the time of readout, this means that each row is exposed during a slightly different time window. The more conventional CCD sensors on the other hand use a *global shutter* (GS), where all pixels are reset simultaneously, and collect light during the same time interval. The downside with a rolling shutter is that since pixels are acquired at different points in time, motion of either camera or target will cause geometrical distortions in the acquired images. Figure 1 shows an example of geometric distortions caused by using a rolling shutter, and the result of our rectification step, as well as two other methods.

Figure 1: Example of rolling shutter imagery. Top left: Frame from an iPhone 3GS camera sequence acquired during fast motion. Top right: Rectification using our rotation method. Bottom left: Rectification using the global affine method. Bottom right: Rectification using the global shift method. Corresponding videos are available on the web [27].

1.1 Related Work

A camera motion between two points in time can be described with a three element translation vector, and a 3 *degrees-of-freedom* (DOF) rotation. For hand-held footage, the rotation component is typically the dominant cause of image plane motion. [31] gave a calculation example of this for the related problem of motion blur during long exposures. (A notable exception where translation is the dominant component is footage from a moving platform, such as a car.) Many new camcorders thus have *mechanical image stabilisation* (MIS) systems that move the lenses (some instead move the sensor) to compensate for small pan and tilt rotational motions (image plane rotations, and large motions, are not handled). The MIS parameters are typically optimised to the frequency range caused by a person holding a camera, and thus work well for such situations. However, since lenses have a certain mass, and thus inertia, MIS has problems keeping up with faster motions, such as caused by vibrations from a car engine. Furthermore, cell-phones, and lower end camcorders lack MIS and recorded videos from these will exhibit RS artifacts.

For cases when MIS is absent, or non-effective, one can instead do post-capture im-

age rectification. There exist a number of different approaches for dealing with special cases of this problem [8, 19, 25, 9, 10, 11]. Some algorithms assume that the image deformation is caused by a globally constant translational motion across the image [8, 25, 11]. After rectification this would correspond to a constant optical flow across the entire image, which is rare in practise. [19] improve on this by giving each row a different motion, that is found by interpolating between constant global inter-frame motions using a Bézier curve. Another improvement is due to [9] and [10]. Here geometric distortion is modelled as a global affine deformation that is parametrised by the scan-line index. Recently [4] improved on [9] and [19] by using not one model per frame, but instead blended linearly between up to 30 affine or translational models across the image rows. This means that their model can cope with motions that change direction several times across a frame. This was made possible by using a high-quality dense optical flow field. They also used a L1 optimisation based on linear programming. However, they still model distortions in the image plane.

All current RS rectification approaches perform warping of individual frames to rectify RS imagery. Note that a perfect compensation under camera translation would require the use of multiple images, as the parallax induced by a moving camera will cause occlusions. Single frame approximations do however have the advantage that ghosting artifacts caused by multi-frame reconstruction is avoided, and is thus often preferred in video stabilisation [20].

Other related work on RS images include structure and motion estimation. [14] study the projective geometry of RS cameras, and also describe a calibration technique for estimation of the readout parameters. The derived equations are then used for structure and motion estimation in synthetic RS imagery. [1] demonstrate that motion estimation is possible from single rolling shutter frames if world point-to-point distances are known, or from curves that are known to be straight lines in the world. They also demonstrate that structure and motion estimation can be done from a single stereo pair if one of the used cameras has a rolling shutter [2].

Video stabilisation has a long history in the literature, an early example is [16]. The most simple approaches apply a global correctional image plane translation [16, 9]. A slightly more sophisticated approach is a global affine model. A special case of this is the zoom and translation model used by [10].

A more sophisticated approach is to use rotational models. Such approaches only compensate for the 3D rotational motion component, and neglect the translations, in a similar manner to mechanical stabilisation rigs [32]. Rotational models estimate a compensatory rotational homography, either using instantaneous angular velocity [32] (differential form), or using inter-frame rotations, e.g. represented as unit quaternions [24]. Since only rotations are corrected for, there are no parallax-induced occlusions to consider, and thus single-frame reconstructions are possible.

The most advanced (and computationally demanding) video stabilisation algorithms make use of *structure-from-motion* (SfM). An early example is the quasi-affine SfM explored by [7]. These methods attempt to also correct for parallax changes in the stabilised views. This works well on static scenes, but introduces ghosting when the scene is dynamic, as blending from multiple views is required [20]. A variant of SfM based stabilisation is the content preserving warps introduced by [20]. Here single frames are used in the reconstruction, and geometric correctness is traded for perceptual plausibil-

ity.

Recently [21] presented a new stabilisation approach based on subspace constraints on 2D feature trajectories. This has the advantage that it does not rely on SfM, which is computationally heavy and sensitive to rolling shutter artifacts. The algorithm does not explicitly model a rolling shutter, instead it is treated as noise. The new algoritm can deal with rolling shutter wobble from camera shake, but not shear introduced by a panning motion.

1.2 Contributions

All the previous approaches to rectification of RS video [8, 19, 25, 9, 10, 11, 4] model distortions as taking place in the image plane. We instead model the 3D camera motion using calibrated projective geometry.

In this article, we focus on a distortion model based on 3D camera rotation, since we have previously shown that it outperforms a combined rotation and translation model [12]. In this article, we extend the rotational model to use multiple knots across a frame. This enables the algorithm to detect non-constant motions during a frame capture. We also introduce a scheme for positioning of the knots that makes the optimisation stable.

We demonstrate how to apply smoothing to the obtained rotation sequence to obtain a high quality video stabilisation, at a low computational cost. This results in a stabilisation similar to rotational models previously used on global shutter cameras. However, as we perform the filtering offline, the amount of smoothing can be decided by the user, post capture. This can e.g. be done using a slider in a video editing application running on a cell-phone. We compare our video stabilisation with the methods in iMovie and Deshaker.

We also introduce a rectification technique using forward interpolation. Many new smartphones have hardware for graphics acceleration, e.g. using the *OpenGL ES 2.0* application programming interface. Such hardware can be exploited using our forward interpolation technique, to allow rectification and stabilisation during video playback.

We recently introduced the first dataset for evaluation of algorithms that rectify rolling shutter video [27]. We now extend it with another 2 sequences, and add a more sophisticated comparison between rectifications and ground truth. Using the dataset, we compare our own implementations of the global affine model [9], and the global shift model [11] to the new method that we propose. Our dataset, evaluation code and supplementary videos are available for download at [27].

1.3 Overview

This article is organised as follows: Section 2, describes how to calibrate a rolling-shutter camera. Section 3 introduces the model and cost function for camera ego-motion estimation. Section 4 discusses interpolation schemes for rectification of rolling-shutter imagery. Section 5 describes how to use the estimated camera trajectory for video stabilisation. Section 6 describes the algorithm complexity and cell-phone implementation feasibility. Section 7 describes our evaluation dataset. In section 8 we use our dataset to compare different rectification strategies, and to compare our 3D rotation model to our own implementations of [9] and [11]. We also compare our stabilisation to

iMovie and Deshaker. In section 9 we describe the supplemental material and discuss the performance of the algorithms. The article concludes with outlooks and concluding remarks in section 10. Appendix A describes the calibration procedure, and appendix B lists the online resources.

2 Rolling Shutter Camera Calibration

In this article, we take the *intrinsic camera matrix*, the *camera frame-rate* and the *inter-frame delay* to be given. This reduces the number of parameters that need to be estimated on-line, but also requires us to calibrate each camera before the algorithms can be used.

On camera equipped cell-phones, such calibration makes good sense, as the parameters stay fixed (or almost fixed, in the case of variable focus cameras) throughout the lifetime of a unit. We have even found that transferring calibrations between cell-phones of the same model works well.

2.1 Geometric Calibration

A 3D point, $\mathbf{X}$, and its projection in the image, $\mathbf{x}$, given in homogeneous coordinates, are related according to

$$\mathbf{x} = \mathbf{K}\mathbf{X} \ , \text{ and } \quad \mathbf{X} = \lambda \mathbf{K}^{-1}\mathbf{x} \ , \tag{1}$$

where $\mathbf{K}$ is a 5DOF upper triangular 3×3 *intrinsic camera matrix*, and λ is an unknown scaling [18].

We have estimated $\mathbf{K}$ for a number of cell-phones using the calibration plane method [33] as implemented in OpenCV.

Note that many high-end cell-phones have variable focus. As this is implemented by moving the single lens of the camera back and forth, the camera focal length will vary slightly. On e.g. the iPhone 3GS the field-of-view varies with about 2°. However, we have found that such small changes in the $\mathbf{K}$ matrix makes no significant difference in the result.

2.2 Readout Time Calibration

The RS chip frame period $1/f$ (where f is the *frame rate*) is divided into a *readout time* t_r, and an *inter-frame delay*, t_d as

$$1/f = t_r + t_d \, . \tag{2}$$

The readout time can be calibrated by imaging a flashing light source with known frequency [14], see appendix A for details. For a given frame rate f, the inter-frame delay can then be computed using (2). For our purposes it is preferable to use rows as fundamental unit, and express the inter-frame delay as a number of *blank rows*:

$$N_b = N_r t_d/(1/f) = N_r(1 - t_r f) \, . \tag{3}$$

Here N_r is the number of image rows.

3 Camera Motion Estimation

Our model of camera motion is a rotation about the camera centre during frame capture, in a smooth, but time varying way. Even though this model is violated across an entire video clip, we have found it to work quite well if used on short frame intervals of 2-4 frames [12]. We represent the model as a sequence of rotation matrices, $\mathbf{R}(t) \in \mathrm{SO}(3)$.

Two homogeneous image points $\mathbf{x}$, and $\mathbf{y}$, that correspond in consecutive frames, are now expressed as:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(N_x)\mathbf{X} \ , \text{ and } \quad \mathbf{y} = \mathbf{K}\mathbf{R}(N_y)\mathbf{X} \tag{4}$$

where N_x and N_y correspond to the time parameters for point $\mathbf{x}$ and $\mathbf{y}$ respectively. This gives us the relation:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(N_x)\mathbf{R}^T(N_y)\mathbf{K}^{-1}\mathbf{y} \, . \tag{5}$$

The time parameter is a linear function of the current image row (i.e. x_2/x_3 and y_2/y_3). Thus, by choosing the unit of time as image rows, and time zero as the top row of the first frame, we get $N_x = x_2/x_3$ for points in the first image. In the second image we get $N_y = y_2/y_3 + N_r + N_b$, where N_r is the number of image rows in a frame, and N_b is defined in (3).

Each correspondence between the two views, (5) gives us two equations (after elimination of the unknown scale) where the unknowns are the rotations. Unless we constrain the rotations further, we now have six unknowns (a rotation can be parametrised with three parameters) for each correspondence. We thus parametrise the rotations with an interpolating linear spline with a number of knots placed over the current frame window, see figure 2 for an example with three frames and $M = 6$ knots. Intermediate rotations are found using spherical linear interpolation [29].

As we need a reference world frame, we might as well fixate that to the start of frame 1, i.e. set $\mathbf{R}_1 = \mathbf{I}$. This gives us $3(M-1)$ unknowns in total for a group of M knots.

3.1 Motion Interpolation

Due to the periodic structure of $\mathrm{SO}(3)$ the interpolation is more complicated than regular linear interpolation.

We have chosen to represent rotations as three element vectors, $\mathbf{n}$, where the magnitude, ϕ, corresponds to the rotation angle, and the direction, $\hat{\mathbf{n}}$, is the axis of rotation, i.e. $\mathbf{n} = \phi\hat{\mathbf{n}}$. This is a minimal parametrisation of rotations, and it also ensures smooth variations, in contrast to e.g. Euler angles. It is thus suitable for parameter optimisation. The vector $\mathbf{n}$ can be converted to a rotation matrix using the matrix exponent, which for a rotation simplifies to Rodrigues formula:

$$\mathbf{R} = \mathrm{expm}(\mathbf{n}) = \mathbf{I} + [\hat{\mathbf{n}}]_x \sin\phi + [\hat{\mathbf{n}}]_x^2(1-\cos\phi) \tag{6}$$

$$\text{where } [\hat{\mathbf{n}}]_x = \frac{1}{\phi}\begin{pmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{pmatrix} . \tag{7}$$

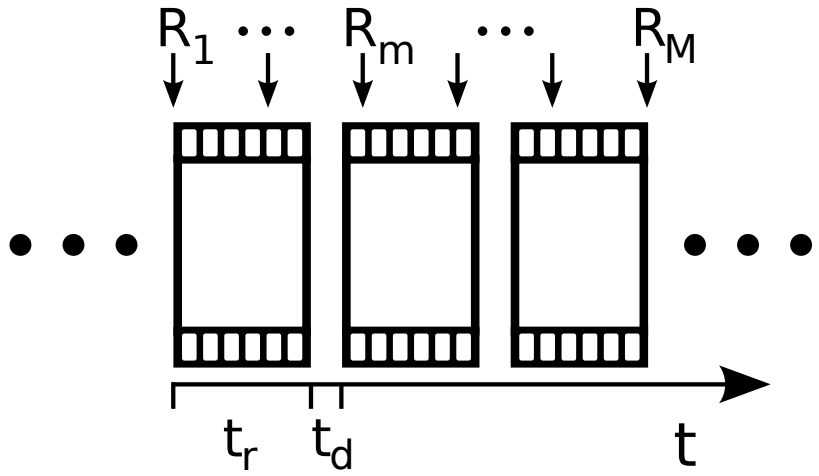


Figure 2: Rotations, $\mathbf{R}_2, \mathbf{R}_3, \ldots, \mathbf{R}_M$ found by non-linear optimisation. Intermediate rotations are defined as interpolations of these, and $\mathbf{R}_1 = \mathbf{I}$. Readout time t_r, and inter-frame delay t_d are also shown.

Conversion back to vector form is accomplished through the matrix logarithm in the general case, but for a rotation matrix, there is a closed form solution. We note that two of the terms in (6) are symmetric, and thus terms of the form $r_{ij} - r_{ji}$ will come from the anti-symmetric part alone. Conversely the trace is only affected by the symmetric parts. This allows us to extract the axis and angle as:

$$\mathbf{n} = \text{logm}(\mathbf{R}) = \phi\hat{\mathbf{n}}\,, \text{ where } \begin{cases} \tilde{\mathbf{n}} = \begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix} \\ \phi = \tan^{-1}(||\tilde{\mathbf{n}}||, \text{tr}\mathbf{R} - 1) \\ \hat{\mathbf{n}} = \tilde{\mathbf{n}}/||\tilde{\mathbf{n}}||\,. \end{cases} \tag{8}$$

It is also possible to extract the rotation angle from the trace of $\mathbf{R}$ alone [26]. We recommend (8), as it avoids numerical problems for small angles. Using (6) and (8), we can perform SLERP (Spherical Linear intERPolation) [29] between two rotations $\mathbf{n}_1$ and $\mathbf{n}_2$, using an interpolation parameter $\tau \in [0, 1]$ as follows:

$$\mathbf{n}_{\text{diff}} = \text{logm}\left(\text{expm}(-\mathbf{n}_1)\text{expm}(\mathbf{n}_2)\right) \tag{9}$$

$$\mathbf{R}_{\text{interp}} = \text{expm}(\mathbf{n}_1)\text{expm}(\tau\mathbf{n}_{\text{diff}}) \tag{10}$$

3.2 Optimisation

We now wish to solve for the unknown motion parameters, using iterative minimisation. For this we need a cost function:

$$J = \epsilon(\mathbf{n}_1, \ldots, \mathbf{n}_N). \tag{11}$$

To this end, we choose to minimise the (symmetric) image-plane residuals of the set of corresponding points $\mathbf{x}_k \leftrightarrow \mathbf{y}_k$:

$$J = \sum_{k=1}^{K} d(\mathbf{x}_k, \mathbf{H}\mathbf{y}_k)^2 + d(\mathbf{y}_k, \mathbf{H}^{-1}\mathbf{x}_k)^2 \tag{12}$$

$$\text{where} \;\; \mathbf{H} = \mathbf{K}\mathbf{R}(\mathbf{x}_k)\mathbf{R}^T(\mathbf{y}_k)\mathbf{K}^{-1} \tag{13}$$

Here the distance function $d(\mathbf{x}, \mathbf{y})$ for homogeneous vectors, is given by:

$$d(\mathbf{x}, \mathbf{y})^2 = (x_1/x_3 - y_1/y_3)^2 + (x_2/x_3 - y_2/y_3)^2 \,. \tag{14}$$

For each frame interval we have M knots for the linear interpolating spline. The first knot is at the top row of the fist frame, and the last knot at the bottom row of the last frame in the interval. The position of a knot, the *knot time* (N_m), is expressed in the unit rows, where rows are counted from the start of the first frame in the interval. E.g. a knot at the top row of the second frame would have the knot time $N_m = N_r + N_b$ and correspond to the rotation $\mathbf{n}_m$.

We denote the evaluation of the spline at a row value N_{curr} by:

$$\mathbf{R} = \texttt{SPLINE}(\{\mathbf{n}_m, N_m\}_1^M \,, N_{\text{curr}}) \,. \tag{15}$$

The value is obtained using `SLERP` as:

$$\mathbf{R} = \texttt{SLERP}(\mathbf{n}_m, \mathbf{n}_{m+1}, \tau) \,, \;\; \text{for} \tag{16}$$

$$\tau = \frac{N_{\text{curr}} - N_m}{N_{m+1} - N_m} \,, \;\; \text{where} \;\; N_m \leq N_{\text{curr}} \leq N_{m+1}. \tag{17}$$

Here `SLERP` is defined in (9,10), N_{curr} is the current row relative to the top row in first frame and N_m, N_{m+1} are the two neighbouring knot times.

For speed, and to ensure that the translation component of the camera motion is negligible, we have chosen to optimise over short intervals of $N = 2$, 3 or 4 frames.

We have chosen to place the knots on different height for the different frames within the frame interval, see figure 3 for two examples. If the knots in two consecutive frames have the same height, the optimisation might drift sideways without increasing the residuals.

There is a simple way to initialise a new interval from the previous one. Once the optimiser has found a solution for a group of frames, we change the origin to the second camera in the sequence (see figure 2). This shift of origin is described by the rotation:

$$\mathbf{R}_{\text{shift}} = \texttt{SPLINE}(\{\mathbf{n}_m, N_m\}_1^M \,, N_r + N_b) \,, \tag{18}$$

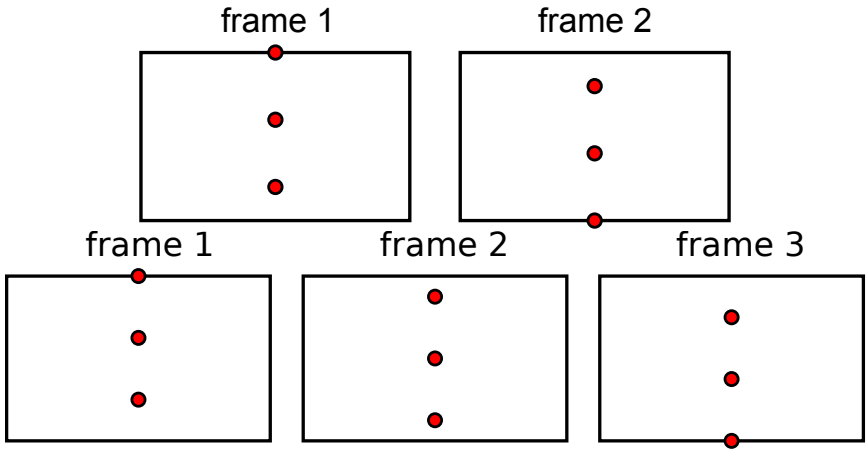


Figure 3: Top: Frame interval of 2 frames with 6 knots. Bottom: Frame interval of 3 frames with 9 knots.

Then we shift the interval one step, by re-sampling the spline knots $\{\mathbf{n}_m\}_1^M$, with an offset of $N_r + N_b$

$$\mathbf{R}_m = \texttt{SPLINE}(\{\mathbf{n}_k, N_k\}_1^M, N_m + N_r + N_b)\,, \tag{19}$$

and finally correct them for the change of origin, using

$$\mathbf{n}'_m = \text{logm}(\mathbf{R}_{\text{shift}}^T \mathbf{R}_m)\,, \quad \text{for } \; m = 1, \ldots, L\,, \tag{20}$$

where N_L is the last time inside the frame interval. As initialisation of the rotations in the newly shifted-in frame, we copy the parameters for the last valid knot, $\mathbf{n}'_L$.

3.3 Point Correspondences

The point correspondences needed to estimate the rotations are obtained through point tracking. We use Harris points [17] detected in the current frame and the chosen points are tracked using the KLT tracker [23, 28]. The KLT tracker uses a spatial intensity gradient search which minimises the Euclidean distance between the corresponding patches in the consecutive frames. We use the scale pyramid implementation of the algorithm in OpenCV. Using pyramids makes it easier to detect large movements.

To increase the accuracy of the point tracker, a crosschecking procedure is used [5]. When the points have been tracked from the first image to the other, the tracking is reversed and only the points that return to the original position (within a threshold) are kept. The computation cost is doubled but outliers are removed effectively.

3.3.1 Correspondence Accuracy Issues

If the light conditions are poor, as is common indoors, and the camera is moving quickly, the video will contain motion blur. This is a problem for the KLT tracker,

which will have difficulties finding correspondences and many tracks will be removed by the crosschecking step.

If the light conditions are good enough for the camera to have a short exposure, and a frame has severe rolling shutter artifacts, the KLT tracker may also have problems finding any correspondences due to local shape distortions. We have however not seen this in any videos, not even during extreme motions. The tracking can on the other hand result in a small misalignment or a lower number of correspondences, when two consecutive frames have very different motions (and thus different local appearances).

It may be possible to reduce the influence of rolling-shutter distortion on the tracking accuracy, by detecting and tracking points again, in the rectified images. This is however difficult because the inverse mapping back to the original image is not always one-to-one and it may thus not be possible to express these new correspondences in the original image grid. How to improve the measurements in this way is an interesting future research possibility.

4 Image Rectification

Once we have found our sequence of rotation matrices, we can use them to rectify the images in the sequence. Each image row has experienced a rotation according to (15). This rotation is expressed relative to the start of the frame, and applying it to the image points would thus rectify all rows to the first row. We can instead align them all to a reference row $\mathbf{R}_{\text{ref}}$ (we typically use the middle row), using a compensatory rotation:

$$\mathbf{R}'(\mathbf{x}) = \mathbf{R}_{\text{ref}}\mathbf{R}^T(\mathbf{x})\,. \tag{21}$$

This gives us a forward mapping for the image pixels:

$$\mathbf{x}' = \mathbf{K}\mathbf{R}_{\text{ref}}\mathbf{R}^T(\mathbf{x})\mathbf{K}^{-1}\mathbf{x} \tag{22}$$

This tells us how each point should be displaced in order to rectify the scene. Using this relation we can transform all the pixels to their new, rectified locations.

We have chosen to perform the rectifying interpolation by utilising the parallel structure of the GPU. A grid of vertices can be bound to the distorted rolling shutter image and rectified with (22) using the *OpenGL Shading Language* (GLSL) vertex shader. On a modern GPU, a dense grid with one vertex per pixel is not a problem, but on those with less computing power, e.g. mobile phones with OpenGL ES 2.0, the grid can be heavily down-sampled without loss of quality. We have used a down-sampling factor of 8 along the columns, and 2 along the rows. This gave a similar computational cost as inverse interpolation on Nvidia G80, but without noticeable loss of accuracy compared to dense forward interpolation.

By defining a grid larger than the input image we can also use the GPU texture out-of-bound capability to get a simple extrapolation of the result.

It is also tempting to use regular, or *inverse interpolation*, i.e. invert (22) to obtain:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(\mathbf{x}')\mathbf{R}_{\text{ref}}^T\mathbf{K}^{-1}\mathbf{x}'\,. \tag{23}$$

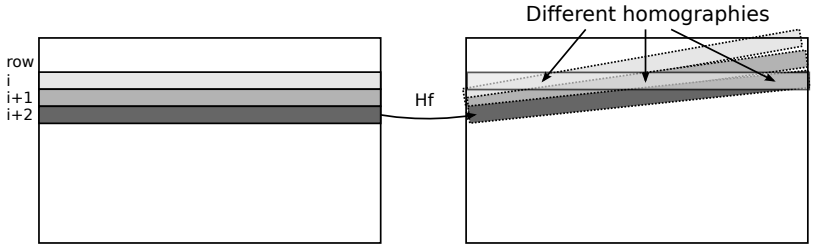


Figure 4: Left: RS distorted image. Right: Output image.

By looping over all values of $\mathbf{x}'$, and using (23) to find the pixel locations in the distorted image, we can cubically interpolate these. If we have large motions this approach will however have problems since different pixels within a row should be transformed with different homographies, see figure 4. Every pixel within a row in the distorted image does however share the same homography as we described in the forward interpolation case. For a comparison between the results of forward and inverse interpolation see figure 5.

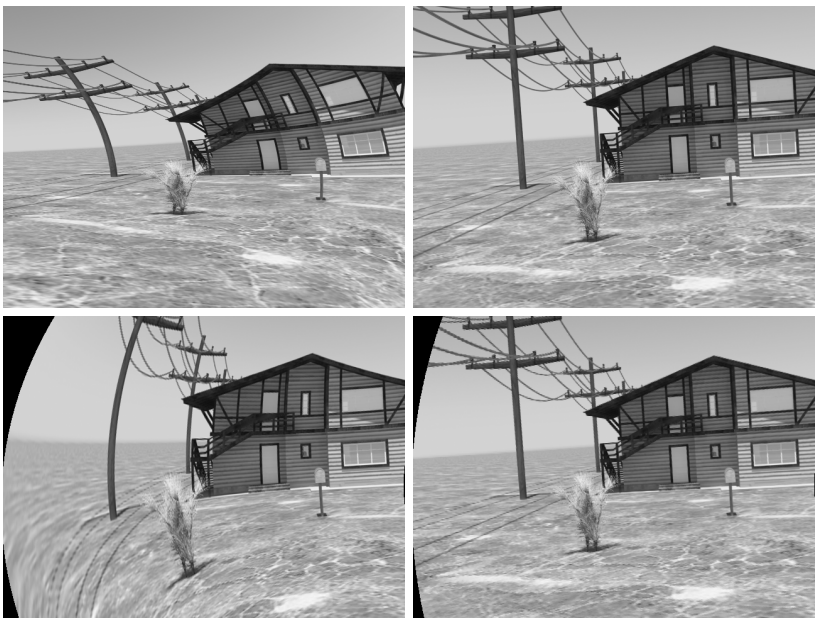


Figure 5: Example of forward and inverse interpolation with large motion. Top left: Rolling shutter frame. Top right: Ground truth. Bottom left: Rectification using inverse interpolation. Bottom right: Rectification using forward interpolation.

5 Video Stabilisation

Our rectification algorithm also allows for video stabilisation, by smoothing the estimated camera trajectory. The rotation for the reference row $\mathbf{R}_{\text{ref}}$ in section 4 is typically set to the current frame's middle row. If we instead do a temporal smoothing of these rotations for a sequence of frames we get a more stabilised video.

5.1 Rotation Smoothing

Averaging on the rotation manifold is defined as:

$$\mathbf{R}^* = \underset{\mathbf{R}\in\text{SO(3)}}{\arg\min} \sum_k d_{\text{geo}}(\mathbf{R}, \mathbf{R}_k)^2 \tag{24}$$

where

$$d_{\text{geo}}(\mathbf{R}, \mathbf{R}_k)^2 = ||\text{logm}(\mathbf{R}_1^T \mathbf{R}_2)||_{\text{fro}}^2 \tag{25}$$

is the *geodesic distance* on the rotation manifold (i.e. the relative rotation angle), $||\cdot||_{\text{fro}}$ is the Fröbenius matrix norm, and logm is defined in (8). Finding $\mathbf{R}$ using (24) is iterative, and thus slow, but [15] showed that the barycentre of either the quaternions or rotation matrices are good approximations.

We have tried both methods, but only discuss the rotation matrix method here. The rotation matrix average is given by:

$$\tilde{\mathbf{R}}_k = \sum_{l=-n}^{n} w_l \mathbf{R}_{k+l} \tag{26}$$

where the temporal window is $2n+1$ and w are weights for the input rotations $\mathbf{R}_k$. We have chosen w_l as a Gaussian filter kernel. To avoid excessive correction at the start and end of the rotation sequence, we extend the sequence by replicating the first (and last) rotation a sufficient number of times.

The output of (26) is not guaranteed to be a rotation matrix, but this can be enforced by constraining it to be a rotation [15]:

$$\begin{aligned} \hat{\mathbf{R}}_k &= \mathbf{U}\mathbf{S}\mathbf{V}^T, \text{ where} \\ \mathbf{U}\mathbf{D}\mathbf{V}^T &= \text{svd}(\tilde{\mathbf{R}}_k), \text{ and } \mathbf{S} = \text{diag}(1, 1, |\mathbf{U}||\mathbf{V}|). \end{aligned} \tag{27}$$

5.2 Common Frame of Reference

Since each of our reference rotations $\mathbf{R}_{\text{ref}}$ (see section 4) has its own local coordinate system, we have to transform them to a common coordinate system before we can apply rotation smoothing to them.

We do this using the $\mathbf{R}_{\text{shift},k}$ matrices in (18) that shift the origin from one frame interval to the next. Using these, we can recursively compute shifts to the absolute coordinate frame as:

$$\mathbf{R}_{\text{abs},k} = \mathbf{R}_{\text{abs},k-1}\mathbf{R}_{\text{shift},k}, \text{ and } \mathbf{R}_{\text{abs},1} = \mathbf{I}. \tag{28}$$

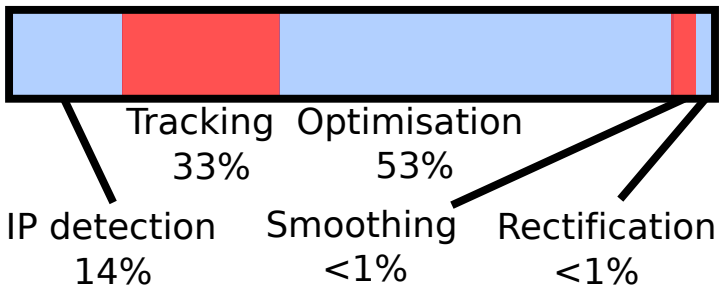


Figure 6: Overview of algorithm complexity. The width of each box roughly corresponds to the fraction of time the algorithm spends there during one pass of optimisation, and one pass of smoothing and playback.

Here $\mathbf{R}_{\text{abs}}$ is the shift to the absolute coordinate system, and $\mathbf{R}_{\text{shift}}$ is the local shift computed in (18).

We can now obtain reference rotations in the absolute coordinate frame:

$$\mathbf{R}'_{\text{ref},k} = \mathbf{R}_{\text{abs},k}\mathbf{R}_{\text{ref},k}\,. \tag{29}$$

After smoothing these, using (26) and (27), we change back to the local coordinate system by multiplying them with $\mathbf{R}^T_{\text{abs},k}$.

6 Algorithm Complexity

The rectification and stabilisation pipeline we have presented can be decomposed into five steps, as shown in figure 6. We have analysed the computational complexity of these steps on our reference platform, which is a HP Z400 workstation, running at 2.66GHz (using one core). The graphics card used in the rectification step is an Nvidia GeForce 8800 GTS.

As can be seen in figure 6, most computations are done in three preprocessing steps. These are executed once for each video clip: (1) Interest-point detection, (2) KLT tracking and crosschecking, and (3) Spline optimisation. We have logged execution times of these steps during processing of a 259 frame video clip, captured with an iPhone 3GS (The stabilisation example video from [12]). This video results in a highly variable number of trajectories (as it has large portions of textureless sky), and thus gives us good scatter plots. We present these timings in figure 7.

The remaining two steps are (4) Rotation smoothing, and (5) Playback of stabilised video. Although these are much less expensive, these steps must be very efficient, as they will be run during interaction with a user when implemented in a cell-phone application.

In the following subsections, we will briefly analyse the complexity of each of the five steps.

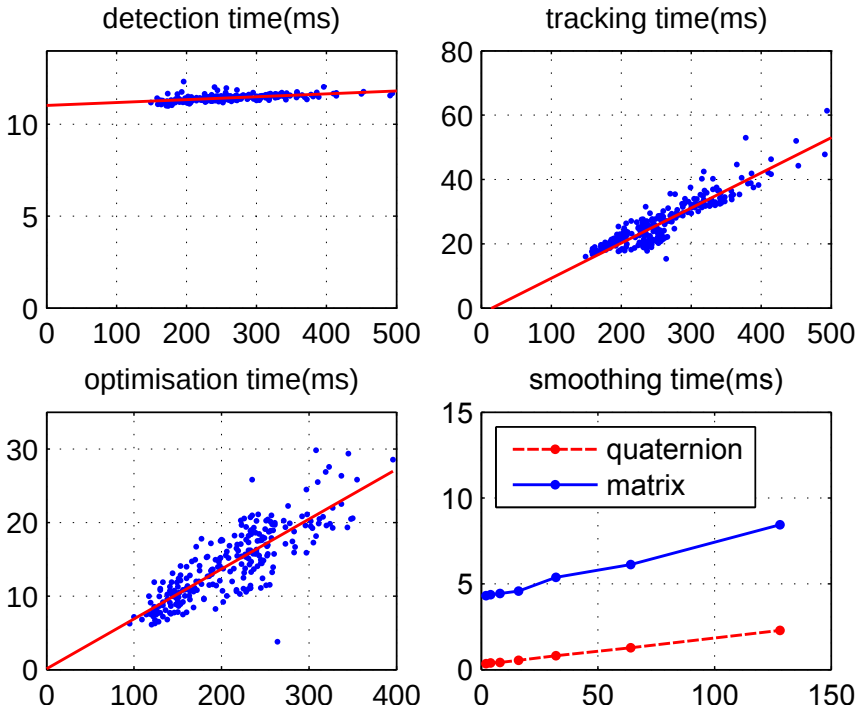


Figure 7: Execution times for algorithm steps. Top left: Detection as function of number of points. Top right: Tracking as function of number of points. Bottom left: Optimisation as function of number of accepted trajectories. Dots are individual measurements, solid lines are least-squares line fits. Bottom right: Rotation smoothing time as function of σ in the Gaussian kernel.

6.1 Interest-Point Detection

The time consumption for the Harris interest point detection is fairly insensitive to the number of points detected. Instead it is roughly linear in the number of pixels. If needed, it can with a small effort be implemented in OpenGL. Average absolute timing: 11.4 ms/frame.

6.2 KLT Tracking

The KLT tracker (see section 3.3) has a time complexity of $O(K \times I)$, where K is the number of tracked points, and I is the number of iterations per point. The average absolute timing for this step is: 26.9 ms/frame including crosschecking rejection. The tracking time as a function of number of interest points is plotted in figure 7.

6.3 Spline Optimisation

The spline optimisation (see section 3.2) has a time complexity of $O(N \times I \times M \times F)$, where N is the number of points remaining after crosschecking rejection, I is the number of iterations, M is the number of knots used in the frame interval, and F is the number of frames in the interval. Average absolute timing: 14.4 ms/frame, when using $M = 3$ and the `levmar` library[1]. The optimisation time as a function of number of accepted trajectories is plotted in figure 7.

6.4 Rotation Smoothing

Rotation smoothing (see section 5.1) is run each time the user changes the amount of stabilisation desired. It is linearly dependent on the length of the video clip, and the kernel size. Figure 7, bottom right, shows a plot of execution times over a 259 frame video clip (averaged over 10 runs).

6.5 Playback of Stabilised Video

The rectification and visualisation step is run once for each frame during video playback. This part is heavily dependent on the graphics hardware and its capability to handle calculations in parallel. The execution time can approximately be kept constant, even though the resolution of the video is changed, by using the same number of vertices for the vertex shader.

6.6 Cell-phone Implementation Feasibility

Our aim is to allow this algorithm to run on mobile platforms. It is designed to run in a streaming fashion by making use of the sliding frame window. This has the advantage that the number of parameters to be estimated is kept low together with a low requirement of memory, which is limited on mobile platforms.

[1] http://www.ics.forth.gr/∼lourakis/levmar/

The most time-consuming part of our algorithm is usually the non-linear optimisation step when optimising for many knots. Our sliding window approach does however enable us to easily initialise the new interval from the previous one. Since cell-phone cameras have constant focal length we do not need to optimise for this. The KLT tracking step also takes a considerable amount of the total time. The time for both tracking and optimisation can however be regulated by changing the Harris detector to select fewer points.

The most time critical part is the stabilisation and the visualisation, because when the camera motion has been estimated, the user wants a fast response when changing the stabilisation settings. The stabilisation part is already fast, and since the visualisation is done on a GPU it is possible to play it back in real-time by down-sampling the vertex grid to match the current hardware capability.

7 Synthetic Dataset

In order to do a controlled evaluation of algorithms for RS compensation we have generated eight test sequences, available at [27], using the Autodesk Maya software package. Each sequence consists of 12 RS distorted frames of size 640×480, corresponding ground-truth *global shutter* (GS) frames, and *visibility masks* that indicate pixels in the ground-truth frames that can be reconstructed from the corresponding rolling-shutter frame. In order to suit all algorithms, the ground-truth frames and visibility masks come in three variants: for rectification to the time instant when the first, middle and last row of the RS frame were imaged.

Each synthetic RS frame is created from a GS sequence with one frame for each RS image row. One row in each GS image is used, starting at the top row and sequentially down to the bottom row. In order to simulate an inter-frame delay, we also generate a number of GS frames that are not used to build any of the RS frames. The camera is however still moving during these frames.

We have generated four kinds of synthetic sequences, using different camera motions in a static scene, see figure 8. The four sequence types are generated as follows:

#1 In the first sequence type, the camera rotates around its centre in a spiral fashion, see figure 8 top left. Three different versions of this sequence exist to test the importance of modelling the inter-frame delay. The different inter-frame delays are $N_b = 0$, 20 and 40 *blank rows* (i.e. the number of unused GS frames).

#2 In the second sequence type, the camera makes a pure translation to the right and has an inter-frame delay of 40 blank rows, see figure 8 top right.

#3 In the third sequence type the camera makes an up/down rotating movement, with a superimposed rotation from left to right, see figure 8 bottom left. There is also a back-and-forth rotation with the viewing direction as axis.

#4 The fourth sequence type is the same as the third except that a translation parallel to the image plane has been added, see figure 8 bottom right. There are three different versions of this type, all with different amounts of translation.

Figure 8: The four categories of synthetic sequences. Left to right, top to bottom: #1 rotation only, #2 translation only, # full 3DOF rotation. and #4 3DOF rotation and translation.

For each frame in the ground-truth sequences, we have created *visibility masks* that indicate pixels that can be reconstructed from the corresponding RS frame, see figure 9. These masks were rendered by inserting one light source for each image row, into an otherwise dark scene. The light sources had a rectangular shape that illuminates exactly the part of the scene that was imaged by the RS camera when located at that particular place. To acquire the mask, a global shutter render is triggered at the desired location (e.g. corresponding to first, middle or last row in the RS-frame).

8 Experiments

8.1 Choice of Reference Row

It is desirable that as many pixels as possible can be reconstructed to a global shutter frame, and for this purpose, rectifying to the first row (as done by [11] and [9]), middle row, or last image row make a slight difference. The dataset we have generated allows us to compare these three choices. In figure 10, we have plotted the fraction of visible pixels for all frames in one sequence from each of our four camera motion categories. As can be seen in this plot, reconstruction to the middle row gives a higher fraction of visible pixels for almost all types of motion. This result is also intuitively reasonable,

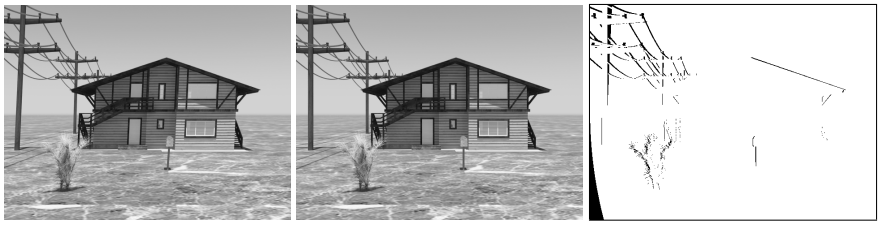


Figure 9: Left to right: Rendered RS frame from sequence of type #2, with $N_b = 40$ (note that everything is slightly slanted to the right), corresponding global-shutter ground-truth, and visibility mask with white for ground-truth pixels that were seen in the RS frame.

as the middle row is closer in time to the other rows, and thus more likely to have a camera orientation close to the average.

8.2 Rectification Accuracy

We have compared our methods to the *global affine model* (GA) [9], and the *global shift model* (GS) [11] on our synthetic sequences, see section 7.

8.2.1 Contrast Invariant Error Measure

When we introduced the RS evaluation dataset [12] we made use of a thresholded Euclidean colour distance to compare ground-truth frames with the rectification output. This error measure has the disadvantage that it is more sensitive in high-contrast regions, than in regions with low contrast. It is also overly sensitive to sub-pixel rectification errors. For these reasons, we now instead use a variance-normalised, error measure:

$$\epsilon(\mathbf{I}_{\text{rec}}) = \sum_{k=1}^{3} \frac{(\mu_k - I_{\text{rec},k})^2}{\sigma_k^2 + \varepsilon\mu_k^2} \,. \tag{30}$$

Here μ_k and σ_k are the means and standard deviations of each colour band in a small neighbourhood of the ground truth image pixel (we use a 3×3 region), and ε is a small value that controls the amount of regularisation. We use $\varepsilon = 2.5e - 3$, which is the smallest value that suppresses high ϵ values in homogeneous regions such as the sky.

The error measure (30) is invariant to a simultaneous scaling of the reconstructed image and the ground-truth image, and thus automatically compensates for contrast changes.

8.2.2 Statistical Interpretation of the Error

If we assume that the colour channels are Gaussian and independent, we get $\epsilon \in \mathcal{X}_3^2$, and this allows us to define a threshold in terms of probabilities. E.g. a threshold t that

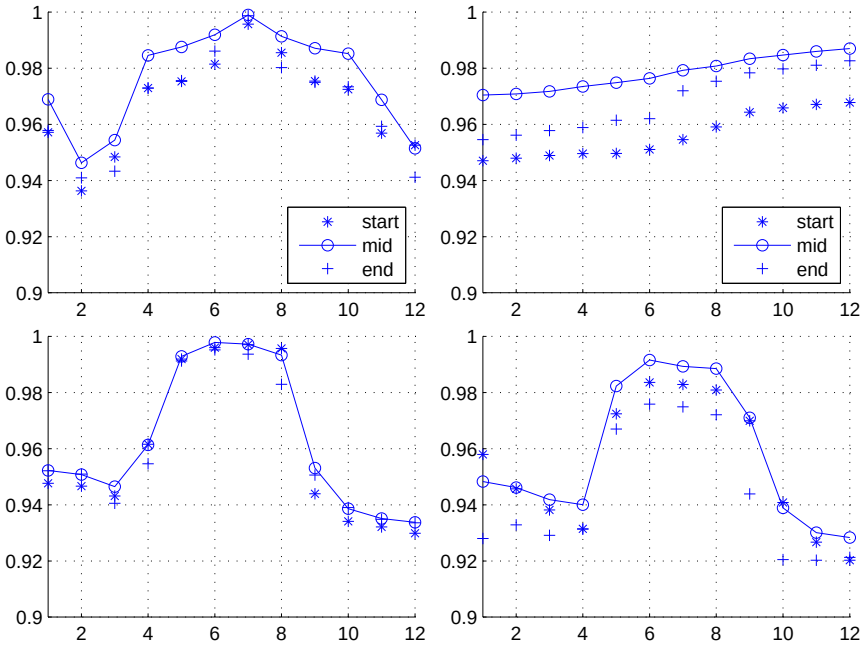


Figure 10: Comparison of visibility masks for sequences #1, #2, #3, and #4 with $B = 40$. Plots show the fraction of non-zero pixels of the total image size.

accepts 75% of the probability mass is found from:

$$0.75 = \int_0^t p_{\chi_3^2}(\epsilon)d\epsilon = \int_0^t \sqrt{\frac{\epsilon}{2\pi}}e^{-\epsilon/2}d\epsilon\,. \tag{31}$$

This results in $t = 4.11$, which is the threshold that we use.

8.2.3 Sub-Pixel Shifts

An additional benefit of using (30) is that the sensitivity to errors due to sub-pixel shifts of the pixels is greatly reduced. We used bicubic resampling to test sub-pixel shifts in the range $\Delta x, \Delta y \in [0.5, 0.5]$ (step size 0.1) on sequence #2. For (30), we never saw more than 0.05% pixels rejected in a shifted image. The corresponding figure for the direct Euclidean distance is 2.5% (using a threshold of 0.3 as in [12]).

8.2.4 Accuracy Measure

As accuracy measure we use the fraction of pixels where the error in (30) is below t. The fraction is only computed within the visibility mask.

For clarity of presentation, we only present a subset of the results on our synthetic dataset. As a baseline, all plots contain the errors for uncorrected frames represented as continuous vertical lines for the means and dashed lines for the standard deviations.

8.2.5 Results

As our reconstruction solves for several cameras in each frame interval, we get multiple solutions for each frame (except in outermost frames). If the model accords with the real motion, the different solutions for the current frame are similar, and we have chosen to present all results for the first reconstruction.

The size of the temporal window used in the optimisation has been studied. The longer the window, the less the rotation-only assumption will hold, and a smaller number of points can be tracked through the frames. In figure 11 the result for two of the sequences can be seen, where the number of frames varied between 2 and 4. In each row, the mean result of all available frames is represented by a diamond centre. Left and right diamond edges indicate the standard deviation.

The result for different numbers of knots can also be seen in figure 11. For fewer knots the optimisation is faster, but the constant motion between sparsely spaced knots is less likely to hold. For many knots, the optimisation time will not only increase (and become more unstable), the probability to have enough good points within the corresponding image region also declines.

On pure rotation scenes it is also interesting to compare the output rotations with the corresponding ground truth. In figure 12 an example from sequence type #3 with four knots spaced over four frames is shown, together with the ground truth rotations. The rotations are represented with the three parameters described in section 3.1. A comparison between different numbers of knots and the ground thruth is shown in figure 13, using the geodesic distance defined in (25).

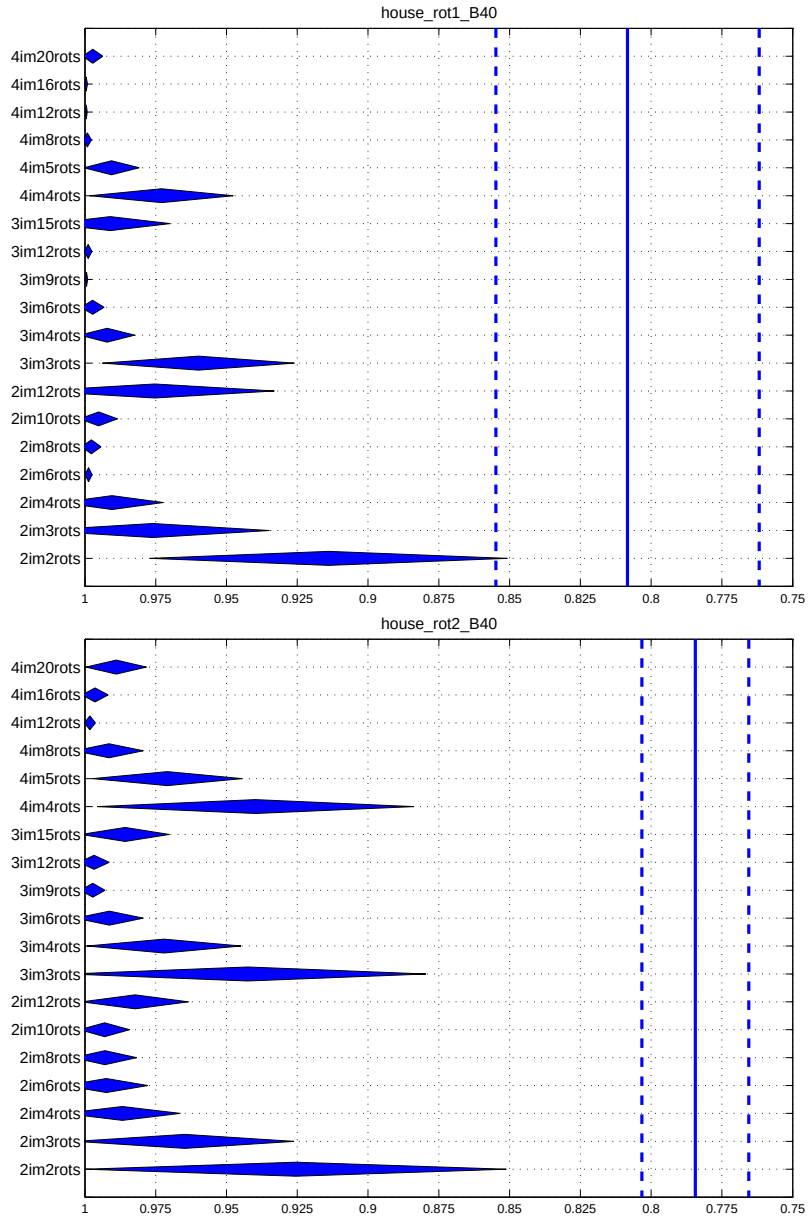


Figure 11: Top: sequence type #1. Bottom: sequence type #3. Two examples of rectification results with different number of knots and window sizes. Diamond centres are mean accuracy, left and right indicate the standard deviation. The continuous line is the unrectified mean and the dashed lines show the standard deviation.

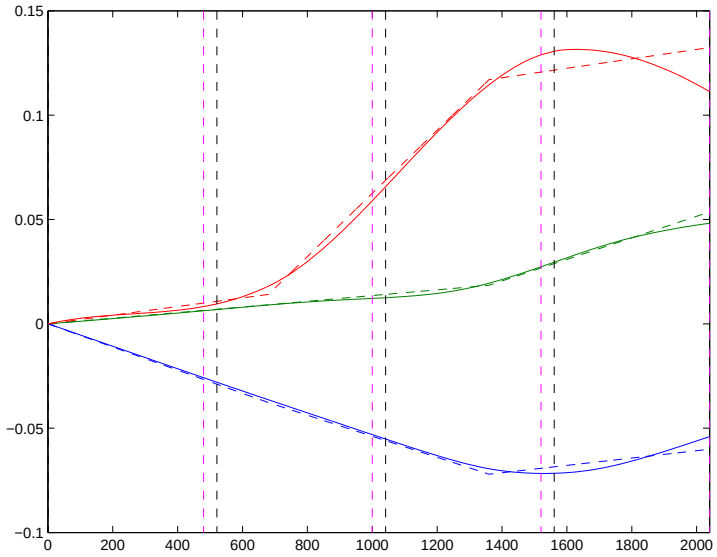


Figure 12: Sequence type #3 with four knots spaced over four frames. Continuous line: ground truth rotations. Dashed lines: result rotations. Vertical black lines define beginning of frames and vertical magenta defines end of frames.

From figure 11 we see that 6 knots over 2 frames, 9 knots over 3 frames and 12 knots over 4 frames give good results. We have also seen good result on real data with these choices and have chosen to do the remaining experiments with the 2 and 3 frame configurations.

For the pure rotation sequences (type #1 and #3) we get almost perfect results, see figures 14 and 15. The GS and GA methods however, do no improvement to the unrectified frames. Figure 16 shows the results from three sequences of type #4 with different amounts of translation. Our methods work best with little translation, but they are still better than the unrectified baseline, as well as the GS and GA methods. In the bottom figure the amount of translation is big and would not be possible to achieve while holding a device in the hand.

An even more extreme case is sequence type # 2, where the camera motion is pure translational. Results for this can be seen in figure 17. This kind of motion only gives rolling shutter effects if the camera translation is fast, e.g. video captured from a car. To better cope with this case, a switching of models can be integrated. The GA method performs better than the GS method on average. This is because the GS method finds the dominant plane in the scene (the house) and rectifies the frame based on this. The ground does not follow this model and thus the rectification there differs substantially from the ground-truth.

Worth noting is that if the rolling shutter effects arise from a translation, objects on different depth from the camera will get different amounts of geometrical distortions. For a complete solution one needs to compensate for this locally in the image. Due to

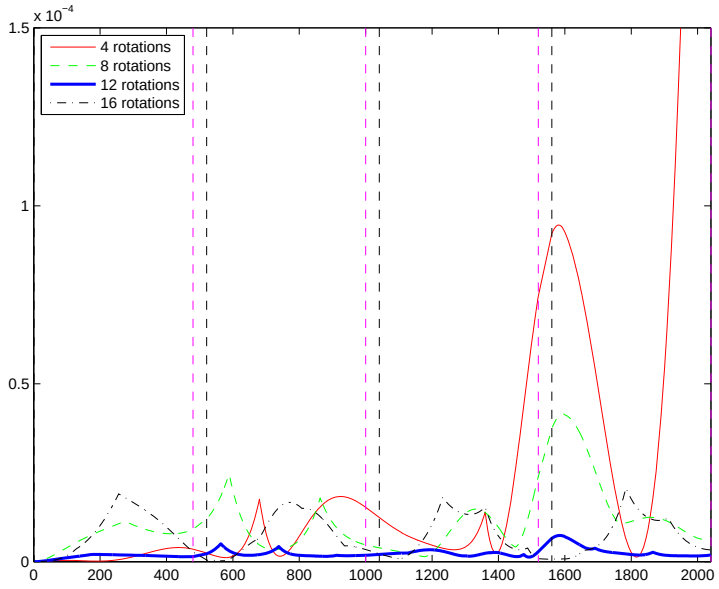


Figure 13: Sequence type #3. Comparison between different numbers of knots and the ground truth rotations. Vertical black lines define beginning of frames and vertical magenta defines end of frames.

occlusions, several frames may also have to be used in the reconstruction.

8.3 Stabilisation of Rolling-Shutter Video

A fair evaluation of video stabilisation is very difficult to make, as the goal is to simultaneously reduce image-plane motions, and to maintain a geometrically correct scene. We have performed a simple evaluation that only computes the amount of image plane motion. In order to see the qualitative difference in preservation of scene geometry, we strongly encourage the reader to also have a look at the supplemental video material.

We evaluate image plane motion by comparing all neighbouring frames in a sequence using the error measure in (30). Each frame thus gets a stabilisation score which is the fraction of pixels that were accepted by (30) over the total number of pixels in the frame. An example of this accuracy computation is given in figure 18.

The comparisons are run on a video clip from an iPhone 3GS, where the person holding the camera was walking forward (Online Resource 1). The walking motion creates noticeable rolling shutter wobble at the end of each footstep.

We compare our results with the following stabilisation algorithms:

#1 Deshaker v 2.5 [30]. This is a popular and free rolling shutter aware video stabiliser. We have set the rolling shutter amount in Deshaker to 92.52% (See appendix A), and consistently chosen the highest quality settings.

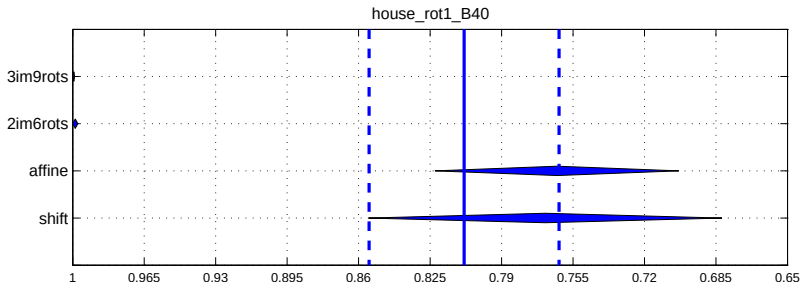


Figure 14: Rectification results for sequence type #1, $N_b = 40$.

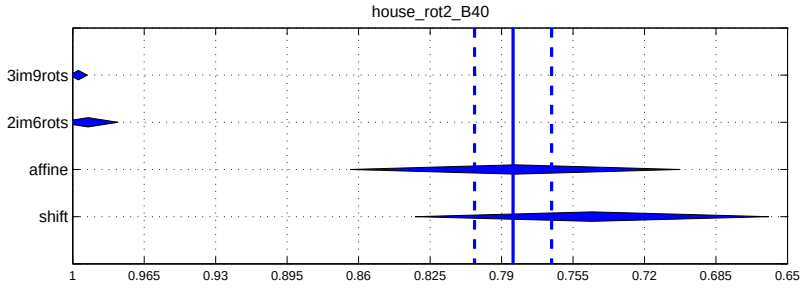


Figure 15: Rectification results for sequence type #3.

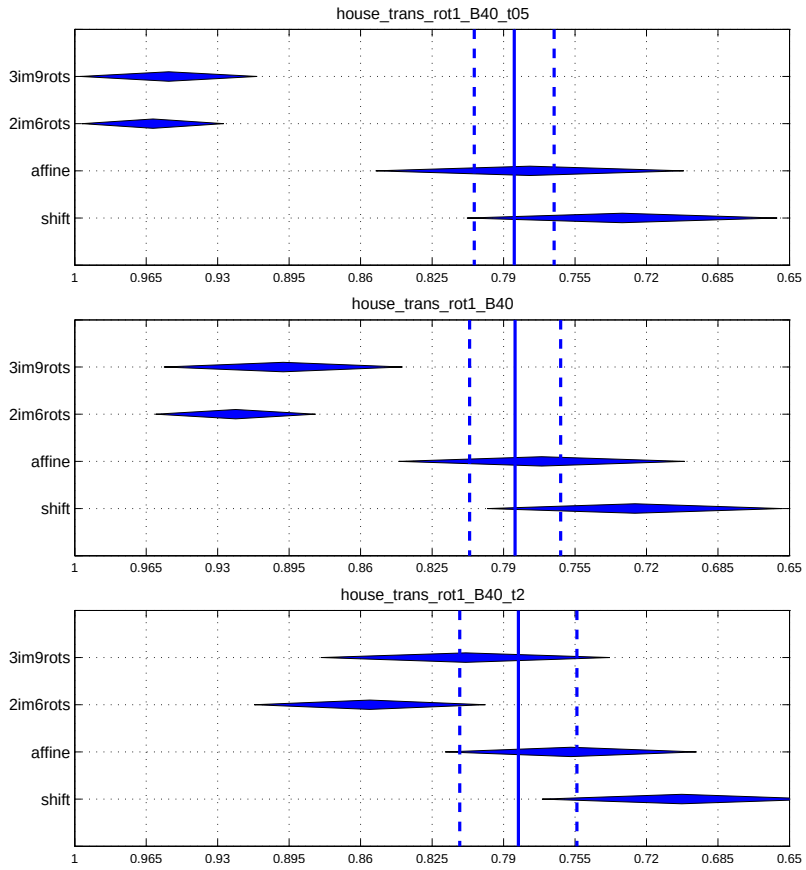


Figure 16: Rectification results for sequences of type #4. Different amounts of translation, starting with least translation at top and most translation at the bottom.

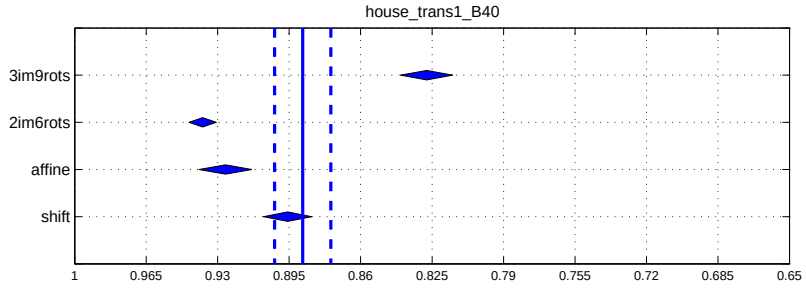


Figure 17: Rectification results for sequence type #2.

Figure 18: Stabilisation accuracy on iPhone 3GS sequence. Left to right: Frame #12, frame #13, accepted pixels (in white).

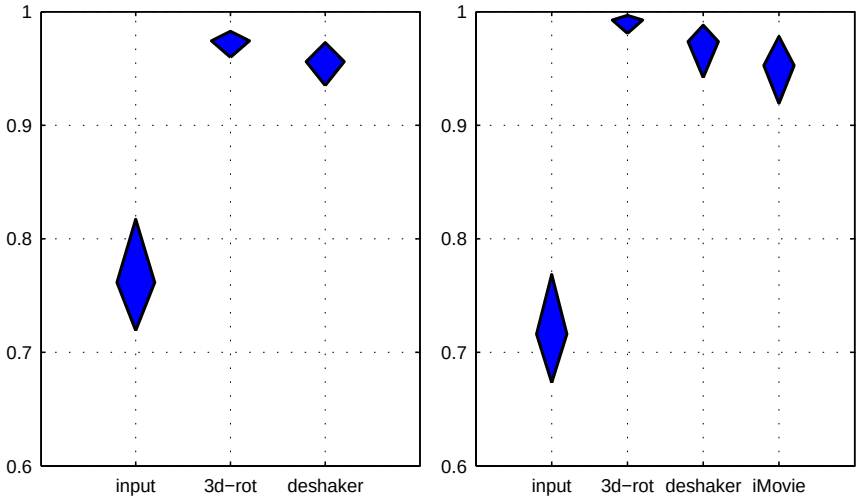


Figure 19: Stabilisation accuracy on iPhone 3GS sequence. Left: stabilisation on original frames. Right: with extrapolation, and zoom to 135%. Diamond centres are median accuracy, tops and bottoms indicate the 25%, and 75% percentiles.

#2 iMovie'09 v8.0.6. [3]. This is a video stabiliser that is bundled with MacOS X. We have set the video to 4:3, 30fps, and stabilisation with the default zoom amount 135%.

The rotation model is optimised over 2 frame windows, with $M = 6$ knots. The stabilised version using a Gaussian filter with $\sigma = 64$ can be seen in Online Resource 3 and with extrapolation and 135% zoom in Online Resource 4.

In figure 19, left we see a comparison with stabilisation in the input grid. This plot shows results from resampling single frames only, borders with unseen pixels remain in these clips (see also figure 18 for an illustration). As no extrapolation is used here, this experiment evaluates stabilisation in isolation. Borders are of similar size in both algorithms, so no algorithm should benefit from border effects.

The second experiment, see figure 19, right, shows results with border extrapolation

turned on, and all sequences are zoomed to 135%. This setting produces video with few noticeable border effects, but at the price of discarding a significant amount of the input video.

The evaluation plots in figure 19, show that our algorithm is better than Deshaker 2.5 at stabilising image plane motion. iMovie'09, which is not rolling-shutter aware, falls significantly behind. However, we emphasise that this comparison tests for image-plane motion only. It is also our impression that the stabilised video has more of a 3D feel to it after processing with the 3D rotation method, than with Deshaker, see Online Resources 4, 6 and 7. The reason for this is probably that Deshaker uses an image plane distortion model.

9 Video examples

In the supplemental material we show results from three videos captured with three different cell-phones, and our result on [22] supplemental video set 1, example 1.

In section 8.3 we used a sequence captured with the iPhone 3GS where Online Resource 1 is the original video, Online Resource 2 is a 135% zoomed version, Online Resource 3 is rectification and stabilisation with our method, Online Resource 4 is our method with additional zoom and extrapolation, Online Resource 5 is the Deshaker result, Online Resource 6 is the Deshaker result with zoom and extrapolation and Online Resource 7 is the result from iMovie.

The supplemental material also contains videos captured from an HTC Desire at 25.87 Hz, see figure 20 left, and a SonyEricsson Xperia X10 at 28.54 Hz, right.

In the HTC Desire sequence the camera is shaken sideways while recording a person, see the original video (Online Resource 8). Online Resource 9 is the result with our rectification and stabilisation method, while Online Resource 10 is stabilisation without rolling-shut-ter compensation. Online Resource 11 and Online Resource 12 are the results from Deshaker and iMovie respectively.

In the SonyEricsson Xperia X10 sequence the person holding the camera is walking while recording sideways, see the original video (Online Resource 13). Online Resource 14 contains the result from our method, Online Resource 15 is the result from Deshaker and Online Resource 16 is the result from iMovie.

From these sequences it is our impression that our 3D rotation model is better at preserving the geometry of the scene. We can also see that Deshaker is significantly better than iMovie which is not rolling shutter aware. Especially on Online Resource 15 we observe good results for Deshaker. Artifacts are mainly visible near the flag poles.

[21] demonstrated their stabilisation algorithm on rolling shutter video. The result can be seen on the supplemental material website, video set 1, example 1 [22]. The algorithm is not rolling shutter aware and distortions are instead treated as noise. The authors report that their algorithm does not handle artifacts like shear introduced by a panning motion but reduces the wobble from camera shake while stabilising the video [21].

Our algorithm needs a calibrated camera, but the parameters are unfortunately not available for this sequence. To obtain a useful $\mathbf{K}$ matrix, the image centre was used as

the projection of the optical centre and different values for the focal length and readout time were tested. As can be seen in Online Resource 17, the 3D structure of the scene is kept whereas the output of [21] still have a a noticeable amount of wobble. The walking up and down motion can still be seen in our result, and this is an artifact of not knowing the correct camera parameters.

Figure 20: Left: First frame from the HTC Desire input sequence. Right: First frame from the SonyEricsson Xperia X10 input sequence.

10 Concluding Remarks

In this article, we have demonstrated rolling-shutter rectification by modelling the camera motion, and shown this to be superior to techniques that model movements in the image plane only. We even saw that image-plane techniques occasionally perform worse than the uncorrected baseline. This is especially true for motions that they do not model, e.g. rotations for the Global shift model [11].

Our stabilisation method is also better than iMovie and Deshaker when we compare the image plane motion. The main advantage of our stabilisation is that the visual appearance of the result videos and the geometry of the scene is much better. Our model also corrects for more types of camera motion than does mechanical image stabilisation (MIS).

In future work we plan to improve our approach by replacing the linear interpolation with a higher order spline defined on the rotation manifold, see e.g. [26]. Another obvious improvement is to optimise parameters over full sequences. However, we wish to stress that our aim is currently to allow the algorithm to run on mobile platforms, which excludes optimisation over longer frame intervals than the 2-4 that we currently use.

For the stabilisation we have used a constant Gaussian filter kernel. For a dynamic video with different kinds of motions, e.g. camera shake and panning, the result would benefit from adaptive filtering.

In general, the quality of the reconstruction should benefit from more measurements. In MIS systems, camera rotations are measured by MEMS gyro sensors [6]. Such sensors are now starting to appear in cell-phones, e.g. the recently introduced

iPhone4. It would be interesting to see how such measurements could be combined with measurements from KLT-track-ing when rectifying and stabilising video.

11 Acknowledgements

We would like to thank SonyEricsson Research Centre in Lund for providing us with a SonyEricsson Xperia X10, and Anders Nilsson at Computer Engineering for lending us the equipment for readout time calibration. This work is funded by the CENIIT organisation at Linköping Institute of Technology, and by the Swedish Research Council.

A Readout Time Calibration

The setup for calibration of readout time is shown in figure 21, top left. An LED is powered from a function generator set to produce a square pulse. The frequency f_o of the oscillation is measured by an oscilloscope. Due to the sequential readout on the RS chip, the acquired images will have horizontal stripes corresponding to on and off periods of the LED, see figure 21, top right. If we measure the period T of the vertical image oscillation in pixels, the readout time can be obtained as:

$$t_r = N_r/(Tf_o)\,, \tag{32}$$

where N_r is the number of image rows, and f_o is the oscillator frequency, as estimated by the oscilloscope.

As can be seen in figure 21 top right, the images obtained in our setup suffer from inhomogeneous illumination, which complicates estimation of T. In their paper, [14] suggest removing the lens, in order to get a homogeneous illumination of the sensor. This is difficult to do on cell-phones, and thus we instead recommend to collect a sequence of images of the flashing LED, and then subtract the average image from each of these. This removes most of the shading seen in the input camera image, see figure 21, middle left, for an example. We then proceed by averaging all columns in each image, and removing their DC. An image formed by stacking such rows from a video is shown in figure 21, middle right. We compute Fourier coefficients for this image along the y-axis, and average their magnitudes to obtain a 1D Fourier spectrum, $F(u)$, shown in figure 21, bottom left.

$$F(u) = \frac{1}{N_f}\sum_{x=1}^{N_f}\left|\frac{1}{N_r}\sum_{y=1}^{N_r} f(y,x)e^{-i2\pi uy/N_r}\right|\,. \tag{33}$$

Here N_f is the number of frames in the sequence. We have consistently used $N_f = 40$. We refine the strongest peak of (33), by also evaluating the function for non-integer values of u, see figure 21, bottom right. The oscillation period is found from the frequency maximum u^*, as $T = N_r/u^*$.

As the Geyer calibration is a bit awkward (it requires a function generator, an oscilloscope and an LED), we have reproduced the calibration values we obtained for a number of different cell-phone cameras in table 1. The "<30" value for frame rate

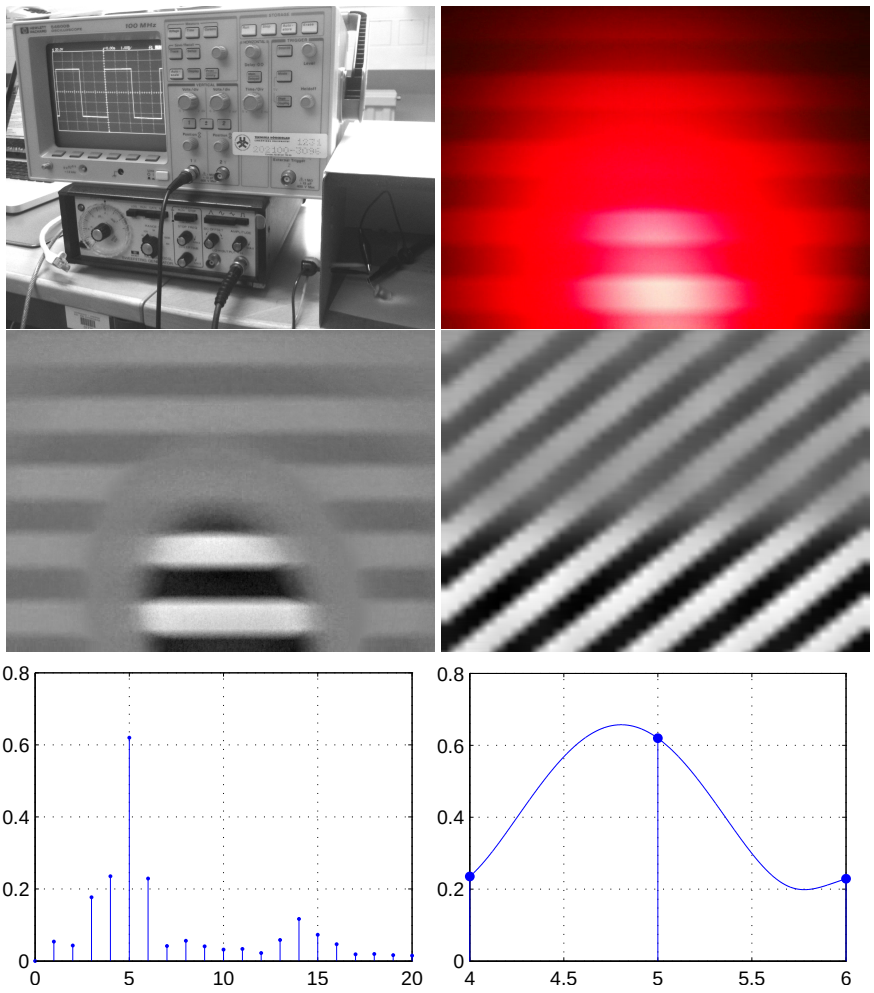


Figure 21: Calibration of a readout time for a rolling-shutter camera. Top left: Used equipment; an oscilloscope, a function generator and a flashing LED. Top right: One image from the sequence. Middle left: Corresponding image after subtraction of the temporal average. Middle right: Image of average columns for each frame in sequence. Bottom left: 1D spectrum from (33), Bottom right: refinement of peak location.

Camera	resolution	f	t_r	N	σ
iPhone 3GS	640×480	30	30.84	7	0.36
iPhone 4(#1)	1280×720	30	31.98	6	0.25
iPhone 4(#2)	1280×720	30	32.37	4	0.27
iPhone 4(#1,front)	640×480	30	29.99	5	0.16
iTouch 4	1280×720	30	30.07	5	0.25
iTouch 4 (front)	640×480	30	31.39	5	0.07
HTC Desire	640×480	< 30	57.84^*	3	0.051
HTC Desire	1280×720	< 30	43.834^*	4	0.034
SE W890i	320×240	14.706	60.78	4	0.16
SE Xperia X10	640×480	< 30	28.386^*	4	0.024
SE Xperia X10	800×480	< 30	27.117^*	5	0.040

Table 1: Calibrated readout times for a selection of cell-phone cameras with rolling shutters. f (Hz) - frame rate reported by manufacturer. t_r (milliseconds) average readout time, N number of estimates, σ standard deviation of t_r measurements. Readout times marked by an "*" are variable. See text for details.

in the table signifies a variable frame rate camera with an upper limit somewhere below 30. The oscillator frequency is only estimated with three significant digits by our oscilloscope, and thus we have averaged obtained readout times for several different oscillator frequencies in order to improve the accuracy.

Reported readout times marked with an "*" are one out of several used by the camera. We have seen that a change of focus, or gain may provoke a change in readout time on those cameras. We have found that rectifications using the listed value look better than those from other readout times, and thus it appears that this value is used most of the time.

The Deshaker webpage [30] reports rolling shutter amounts ($a = t_r \times f$) for a number of different rolling shutter cameras. The only cell-phone currently included is the iPhone4, which is reported as $a = 0.97 \pm 0.02$. Converted to a readout time range, this becomes $t_r \in [31.67\text{ms}, 33\text{ms}]$ which is the range where we find our two iPhone4 units.

B Online Resources

Number	Filename	Description
1	ESM_1.mpg	iPhone 3GS input sequence
2	ESM_2.mpg	iPhone 3GS 135% zoomed input
3	ESM_3.mpg	iPhone 3GS rectification and stabilisation
4	ESM_4.mpg	iPhone 3GS rectification and stabilisation with 135% zoom and extrapolation
5	ESM_5.mpg	iPhone 3GS Deshaker
6	ESM_6.mpg	iPhone 3GS Deshaker with 135% zoom and extrapolation
7	ESM_7.mpg	iPhone 3GS iMovie with 135% zoom and extrapolation
8	ESM_8.mpg	HTC Desire input sequence
9	ESM_9.mpg	HTC Desire rectification and stabilisation
10	ESM_10.mpg	HTC Desire stabilisation only
11	ESM_11.mpg	HTC Desire Deshaker
12	ESM_12.mpg	HTC Desire iMovie with 135% zoom and extrapolation
13	ESM_13.mpg	SonyEricsson Xperia X10 input sequence
14	ESM_14.mpg	SonyEricsson Xperia X10 rectification and stabilisation
15	ESM_15.mpg	SonyEricsson Xperia X10 Deshaker
16	ESM_16.mpg	SonyEricsson Xperia X10 iMovie with 135% zoom and extrapolation
17	ESM_17.mpg	Rectification and stabilisation results on sequence Liu et al. 2011 Supplemental Video Set 1, example 1

References

[1] Omar Ait-Aider, Adrien Bartoli, and Nicolas Andreff. Kinematics from lines in a single rolling shutter image. In *CVPR'07*, Minneapolis, USA, June 2007.

[2] Omar Ait-Aider and Francois Berry. Structure and kinematics triangulation with a rolling shutter stereo rig. In *IEEE International Conference on Computer Vision*, 2009.

[3] Apple Inc. iMovie'09 video stabilizer, 2010. http://www.apple.com/ilife/imovie/.

[4] Simon Baker, Eric Bennett, Sing Bing Kang, and Richard Szeliski. Removing rolling shutter wobble. In *IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco, USA, June 2010. IEEE Computer Society, IEEE.

[5] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. In *IEEE International Conference on Computer Vision (ICCV07)*, Rio de Janeiro, Brazil, 2007.

[6] Jonathan Bernstein. An overview of MEMS inertial sensing technology. *Sensors Magazine*, 2003(1), February 2003.

[7] C. Buehler, M. Bosse, and L. McMillan. Non-metric image-based rendering for video stabilization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'01)*, pages 609–614, 2001.

[8] Li-Wen Chang, Chia-Kai Liang, and H.H. Chen. Analysis and compensation of rolling shutter distortion for CMOS image sensor arrays. In *International Symposium on Communications (ISCOM05)*, 2005.

[9] Won-Ho Cho and Ki-Sang Hong. Affine motion based CMOS distortion analysis and CMOS digital image stabilization. *IEEE Transactions on Consumer Electronics*, 53(3):833–841, August 2007.

[10] Won-Ho Cho, Dae-Woong Kim, and Ki-Sang Hong. CMOS digital image stabilization. *IEEE Transactions on Consumer Electronics*, 53(3):979–986, 2007.

[11] Jung-Bum Chun, Hunjoon Jung, and Chong-Min Kyung. Suppressing rolling-shutter distortion of CMOS image sensors by motion vector detection. *IEEE Transactions on Consumer Electronics*, 54(4):1479–1487, 2008.

[12] Per-Erik Forssén and Erik Ringaby. Rectifying rolling shutter video from hand-held devices. In *IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco, USA, June 2010. IEEE Computer Society, IEEE.

[13] Abbas El Gamal and Helmy Eltoukhy. CMOS image sensors. *IEEE Circuits and Devices Magazine*, May/June 2005.

[14] Christopher Geyer, Marci Meingast, and Shankar Sastry. Geometric models of rolling-shutter cameras. In *6th OmniVis WS*, 2005.

[15] Claus Gramkow. On averaging rotations. *International Journal of Computer Vision*, 42(1/2):7–16, 2001.

[16] R.R. Green, H.W. Mahler, and J.E. Siau. Television picture stabilizing system. US Patent 4,403,256, September 1983.

[17] C. G. Harris and M. Stephens. A combined corner and edge detector. In *4th Alvey Vision Conference*, pages 147–151, September 1988.

[18] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[19] Chia-Kai Liang, Li-Wen Chang, and H.H. Chen. Analysis and compensation of rolling shutter effect. *IEEE Transactions on Image Processing*, 17(8):1323–1330, August 2008.

[20] Feng Liu, Michael Gleicher, Hailin Jin, and Aseem Agarwala. Content-preserving warps for 3D video stabilization. In *ACM Transactions on Graphics*, 2009.

[21] Feng Liu, Michael Gleicher, Jue Wang, Hailin Jin, and Aseem Agarwala. Subspace video stabilization. *ACM Transactions on Graphics*, 30(1), January 2011.

[22] Feng Liu, Michael Gleicher, Jue Wang, Hailin Jin, and Aseem Agarwala. Subspace video stabilization, supplemental video set 1, 2011. http://web.cecs.pdx.edu/~fliu/project/subspace_stabilization/.

[23] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI81*, pages 674–679, 1981.

[24] C. Morimoto and R. Chellappa. Fast 3D stabilization and mosaic construction. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR97)*, pages 660–665, San Juan, Puerto Rico, June 1997.

[25] Steven P. Nicklin, Robin D. Fisher, and Richard H. Middleton. Rolling shutter image compensation. In *Robocup 2006 LNAI 4434*, pages 402–409, 2007.

[26] F.C. Park and Bahram Ravani. Smooth invariant interpolation of rotations. *ACM Transactions on Graphics*, 16(3):277–295, July 1997.

[27] Erik Ringaby. Rolling shutter dataset with ground truth, 2010. http://www.cvl.isy.liu.se/research/rs-dataset.

[28] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR'94*, Seattle, June 1994.

[29] Ken Shoemake. Animating rotation with quaternion curves. In *Int. Conf. on CGIT*, pages 245–254, 1985.

[30] Gunnar Thalin. Deshaker video stabilizer plugin v2.5 for VirtualDub, 2010. http://www.guthspot.se/video/deshaker.htm.

[31] Oliver Whyte, Josef Sivic, Andrew Zisserman, and Jean Ponce. Non-uniform deblurring for shaken images. In *IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco, USA, June 2010. IEEE Computer Society, IEEE.

[32] Y. S. Yao, P. Burlina, R. Chellappa, and T. H. Wu. Electronic image stabilization using multiple visual cues. In *International Conference on Image Processing (ICIP'95)*, pages 191–194, Washington DC, USA, October 1995.

[33] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

Chapter C

Scan Rectification for Structured Light Range Sensors with Rolling Shutters

This is an edited version of the paper:

> Erik Ringaby and Per-Erik Forssén. Scan rectification for structured light range sensors with rolling shutters. In *IEEE International Conference on Computer Vision*, Barcelona, Spain, November 2011. IEEE Computer Society.

Scan Rectification for Structured Light Range Sensors with Rolling Shutters

Erik Ringaby and Per-Erik Forssén
Department of Electrical Engineering, Computer Vision Laboratory
Linköping University, Sweden

Abstract

Structured light range sensors, such as the Microsoft Kinect, have recently become popular as perception devices for computer vision and robotic systems. These sensors use CMOS imaging chips with electronic rolling shutters (ERS). When using such a sensor on a moving platform, both the image, and the depth map, will exhibit geometric distortions. We introduce an algorithm that can suppress such distortions, by rectifying the 3D point clouds from the range sensor. This is done by first estimating the time continuous 3D camera trajectory, and then transforming the 3D points to where they would have been, if the camera had been stationary. To ensure that image and range data are synchronous, the camera trajectory is computed from KLT tracks on the structured-light frames, after suppressing the structured-light pattern. We evaluate our rectification, by measuring angles between the visible sides of a cube, before and after rectification. We also measure how much better the 3D point clouds can be aligned after rectification. The obtained improvement is also related to the actual rotational velocity, measured using a MEMS gyroscope.

1 Introduction

Structured light range sensors (SLRS) have recently become popular, e.g. Microsoft Kinect and the Asus WAVI Xtion. Both these devices are based on a patented reference implementation by the company Primesense [17]. The intended purpose for these sensors is full body gesture based human-computer interfaces. As these devices deliver quasi-dense depth maps at 30 fps (computed using a built-in SoC) they also have many other applications (see e.g.
`http://kinecthacks.net/` for some examples).

We want to make these sensors more useful for dynamic robot perception. 3D range sensing has proven to be very useful for autonomous robots, as demonstrated in the 2007 DARPA urban challenge [21]. In this competition, all of the successful contenders made heavy use of 3D sensing, both for mapping, and to perceive nearby obstacles. However, the high price for time-of-flight and laser-range sensors have this far limited a more widespread use.

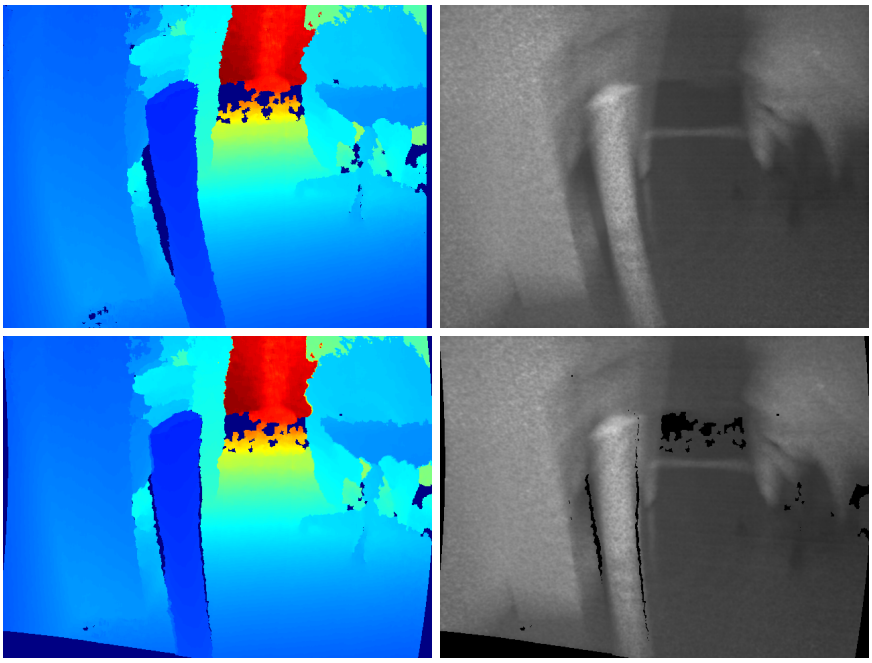


Figure 1: Top row: Depth map, and NIR frame from a Kinect sequence acquired during fast motion. Bottom row: Re-projected depth map, and NIR frame from rectified point cloud. The structured light pattern in the NIR frames have been suppressed using the method in section 3.

SLRS are directly useful for short range obstacle perception (typically up to 3.5m [17]), and they also have the potential to be useful in *simultaneous localisation and mapping* (SLAM) [12]. Doing SLAM using these devices is somewhat problematic, as they make use of CMOS cameras with *electronic rolling shutters* (ERS). In an ERS camera, all pixels are not exposed at the same time (as is the case in a global shutter camera). Instead each row is exposed at a slightly different time interval, that is characterised by the sensor readout time [10, 9, 2]. The effect of this is geometric distortions in the image, when either the camera or the scene changes during exposure, see figure 1, top row.

Unless rolling shutter distortions are modelled, a SLAM system using a rolling shutter sensor is limited to slow motions, or move-stop-look image acquisition cycles (analogous to stop motion animation).

In this paper, we introduce an algorithm that rectifies sensor data affected by rolling shutter distortions. An example of sensor data before, and after the proposed rectification is shown in figure 1.

1.1 Related Work

Simultaneous Localisation and Mapping (SLAM) is the on-line equivalent of the *Structure from Motion* problem [11]. SLAM has been extensively studied over the years, and a good introduction is given in [7]. We do not study the full SLAM problem, instead the algorithm presented in this paper is a pre-processing step that makes the output from SLRS systems useful in a SLAM framework under more general motions.

An alternative to our proposed correction is to use conventional global-shutter sensors, as has been explored by others. Se and Jasiobedzki built a structured light stereo system using a global shutter stereo camera [18]. In order to allow dense correspondences from stereo in untextured environments, a random dot *structured light pattern* (SLP) is projected onto the scene in every second frame. Frames without the SLP are used to acquire textures for the 3D meshes, and to compute SIFT feature correspondences that are used to align the 3D meshes over longer pose changes.

For SLRS systems with rolling shutters, we have only found one attempt at SLAM. Henry et al. [12] have used the Primesense reference platform [17] to do 3D mapping. 3D point clouds are aligned in two steps, first a rigid body transformation between two scans is estimated using visual features. This is then refined using a fusion of visual feature alignment, and range scan alignment from the *Iterative Closest Point algorithm* (ICP) [23]. This solution appears to be effective as loop-closing is used to correct for drifts, but as mentioned on the author's YouTube channel (RGBDvision) it can fail "in the hands of a non technical user", and a requirement is that care is taken to avoid fast camera motions.

There has also been work on rectification of sweeps from moving laser range sensors, where the sensor is rotating [5], or nodding [8]. These are related as they also deal with rectification of 3D point clouds. In [8] planes are found in the scene, and then 3D rotations are optimised for to make the planes as flat as possible. In [5] ICP between neighbouring scans is used, augmented with local scene structure constraints. The sought trajectory is first gridded, and then interpolated with a cubic spline.

Ait-Aider et al. [1] and Klein et al. [13] solve the *perspective-n-point problem* (PnP) [11] under linear motion across a rolling shutter frame. This is somewhat similar to our problem, but note that we have distorted 3D to 3D correspondences, whereas [1, 13] deal with 2D to 3D correspondences, where the 3D points are assumed undistorted.

Another related line of work is rectification of rolling shutter video. This problem has been studied, and solved to some extent [9, 2]. What is different here is that in SLRS systems we also have access to depth values in most pixels, and these allow us to robustly solve for the full 3D camera trajectory, instead of resorting to affine motion [2], or rotation only models [9].

Our rolling shutter model is similar to [9], but instead of assuming a pure rotational motion, or that the scene is purely planar, we model both the camera rotation and translation in an arbitrary static scene. This is possible since we have the depth values for most pixels in the image, and can formulate a cost function on the 3D point correspondences.

1.2 Contributions

We introduce a scheme for scan-matching on SLRS with rolling shutters (e.g. the Microsoft Kinect and the Asus Wavi Xtion), under more general camera motions.

- We describe a simple and efficient technique to remove the structured light pattern from the NIR images. This allows us to use feature tracking in the NIR images.
- We describe how to automatically tune the parameters of the structured light filter, to maximise the performance of the feature tracking step.
- We provide estimates of the readout times for both the NIR and the colour cameras on the Kinect. These should be useful for any researcher that wants to use the Kinect under a rolling shutter model.
- We derive models of rolling shutter correction of structured light range scans, and verify their effectiveness on real data.
- We demonstrate the effectiveness of our approach using experiments on real data.

1.3 Example Calculation

A reasonable question to ask is when the rolling shutter problem matters. To answer this, we give an example calculation for a panning Kinect NIR sensor below (the calculation for tilt is similar, but translations are more complicated characterise, as the distortion now depends on the distance to scene objects). For a pin-hole camera, the sensor width w in pixels, the horizontal field of view h_{FoV}, and the focal length f are related according to [11]:

$$w/(2f) = \tan(h_{\text{FoV}}/2)\,. \tag{1}$$

Assume that we can tolerate a pixel distortion of 5 pixels between the top and the bottom of the frame. For $h_{\text{FoV}} = 58^\circ$ and $w = 640$ [17] we get the focal length $f = 577.3$ pixels. If we now set $w = 5$, and solve for h_{FoV} we get the rotation corresponding to a 5 pixel skew at the image centre (this is an underestimate, as pixels in the periphery span smaller angles). The result is $\theta = 0.496^\circ$. By dividing this by the readout time $r = 30.55$ msec for the Kinect NIR camera (see section 5), we get the angular velocity, $\omega = \theta/r = 16.2^\circ/\text{sec}$.

That is, if we want to bound the geometric distortion to 5 pixels, we need to constrain the camera pan to always be below $16.2^\circ/\text{sec}$ (At this speed a full 360° pan lasts 22.2 seconds). For tilt we instead get a $14.2^\circ/\text{sec}$ bound.

1.4 Overview

This paper is organised as follows: Section 2 briefly describes our approach to solve the range scan rectification problem. In section 3 we introduce a filter that suppresses the structured light pattern, and demonstrate how to tune the filter for maximal feature tracking performance. In section 4 we introduce the motion model and cost functions for camera ego-motion estimation, and methods for rectification of point clouds, depth maps and video frames. Section 5 describes how the sensor calibration was performed.

In section 6 we evaluate our algorithm by measuring angles between the visible sides of a cube, and by comparing the closest point distances in the rectified point clouds to the unrectified point clouds. The paper concludes with outlooks and concluding remarks in section 7.

2 Our Approach

As both the NIR structured light camera and the RGB camera have rolling shutters, and their readout times are different in general, the correspondence problem is problematic under camera motion. (Corresponding pixels in the NIR and RGB images are in general acquired at slightly different times, as the acquisition time depends on the image row.) We use only the NIR camera here, as this ensures temporal correspondence between depth and intensity values. We have also noticed that indoors, the NIR camera uses shorter shutter speeds than the RGB camera, resulting in less motion blur. Using the NIR camera images does however require that we can somehow suppress the influence of the structured light pattern present in the NIR images. Our full approach consists of the following steps:

1. We suppress the structured light pattern in the NIR images, and use feature tracking to find correspondences between frames.
2. We solve the ego-motion estimation problem by optimising over the continuous NIR camera trajectory to minimise the alignment errors in the 3D model.
3. We rectify the 3D model using the estimated motion, and rectify both the filtered NIR images, and the depth map, by projecting the rectified point cloud through the camera again.

3 Removing the Structured Light Pattern

The structured light pattern consists of small circular disks of uniform illumination (see figure 2, left). We have noticed that the scene in-between these illuminated dots is also illuminated, by a weak ambient light. In order to track regions in the NIR images, we thus remove the structured light pattern, and use interpolation to fill in the gaps. We detect the structured light pattern peaks using normalised difference of Gaussian filtering of the NIR image $f(\mathbf{x})$:

$$s(\mathbf{x}) = \frac{(f * (g_{\sigma_1} - g_{\sigma_2}))(\mathbf{x})}{(f * g_{\sigma_2})(\mathbf{x})} \,. \tag{2}$$

Here g_{σ_1} and g_{σ_2} are two Gaussian kernels, with $\sigma_1 < \sigma_2$, and '$*$' is the convolution operator. Note that since the Gaussian kernel is separable, and f can be factored in, the whole operation consists only of four 1D convolutions, and some point-wise operations.

The numerator in (2) serves as a pattern-detector, while the denominator is a local magnitude normalisation. The normalisation is useful because the structured light pattern varies substantially in magnitude across a scene.

In order to remove the structured light pattern, we make use of a technique called *normalized averaging* [16, 14]. This requires that we first convert the pattern function to a *confidence map* $c \in [0, 1]$:

$$c(\mathbf{x}) = \max(0, \min(1, s(\mathbf{x}) \cdot w)) \,, \tag{3}$$

where w is a parameter that controls the scaling of the input. Using the confidence signal, we can now remove the structured light pattern using the quotient of two additional convolutions:

$$\hat{f}(\mathbf{x}) = \frac{(f \cdot c * g_{\sigma_3})(\mathbf{x})}{(c * g_{\sigma_3})(\mathbf{x})} \,, \tag{4}$$

where the σ_3 parameter controls the amount of blurring in the output image. The result of this operation is shown in figure 2, right. Note that the input image has also been gamma corrected by taking the square root of image intensities in the range $f \in [0, 1]$.

Figure 2: Removal of structured-light pattern in NIR image. Left: input image (gamma corrected) Right: result of pattern removal.

3.1 Parameter Tuning

We tune the parameters for the SLP filter to give the best performance for the pyramid KLT tracker [6] in OpenCV. We do this by first selecting five pairs of images in a sequence where the camera was rotating sideways. We then run the SLP filter with a particular parameter setting. Next we detect interest points using the *good features to track measure* [19], and run the KLT tracker on these points. To remove bad tracks, we also apply the a crosschecking rejection with a threshold of 0.5 pixels [9, 3]. After this we have a set of correspondences: $\{\mathbf{x}_k \leftrightarrow \mathbf{y}_k\}_1^K$, where $\mathbf{x}_k$ are interest points, and $\mathbf{y}_k$ are the corresponding locations found by KLT. As we know that the camera was panning, we then separate the correspondences into two categories:

$$\begin{aligned} \mathcal{G} &= \{\mathbf{x}_k \leftrightarrow \mathbf{y}_k : \mathbf{x}_{k,1} - \mathbf{y}_{k,1} \leq -5, |\mathbf{x}_{k,2} - \mathbf{y}_{k,2}| \leq 10\} \\ \mathcal{B} &= \{\mathbf{x}_k \leftrightarrow \mathbf{y}_k : k \in [1, K]\} \setminus \mathcal{G} \end{aligned} \tag{5}$$

where the thresholds for $\mathcal{G}$ membership were found by manual inspection. We can now define a score function as

$$J(\sigma_1, \sigma_2, w, \sigma_3) = |\mathcal{G}|/(|\mathcal{G}| + |\mathcal{B}|)\,. \tag{6}$$

That is, we want to maximise the fraction of correct correspondences.

The simple form of a box constraint in (5) has been chosen as this test needs to be performed in the centre of an optimisation loop. In practise we have found it to be quite effective at finding good filter parameters. Note also in particular that we can not use e.g. a fundamental matrix, or homography constraint [11] for outlier rejection here, as the camera motion causes each image row in a rolling shutter camera to move in a different way.

We have used coordinate-wise optimisation, with alternating exhaustive search in a fixed grid along each coordinate. This is a simple and reasonably efficient way to find useful parameters, it is also straight-forward to implement. Note however that there is no guarantee that the global minimum is found, and that more sophisticated approaches exist. Note also that e.g. gradient descent, and Newton style methods are not suitable, as the score function (6) is not continuous. The parameters we found are listed in table 1. As 5×5 blocks gave the highest score, we have used this setting throughout the paper.

block size	σ_1	σ_2	w	σ_3	J
13×13	0.35	1.0	4.5	3.5	0.8909
11×11	0.475	1.0	4.5	3.4	0.8959
9×9	0.575	2.4	5.0	3.4	0.8991
7×7	0.375	2.1	5.0	3.4	0.8989
5×5	0.575	2.3	9.0	3.4	**0.9026**
3×3	0.575	2.4	9.0	3.4	0.8887

Table 1: Parameters at maximal scores found for different KLT block sizes.

4 Geometry Estimation and Rectification

Since the sensor has a rolling shutter, the reconstructed 3D scene will have geometric distortions if the sensor is moving during a frame capture. To compensate for this we need to model the camera motion and rectify the 3D points accordingly. In contrast to previous work that assumed affine [2], or purely rotational models [9], we model both the camera rotation and translation in an arbitrary static scene. This is possible since we have the depth values for most pixels in the image, and can formulate a cost function on the 3D point correspondences.

4.1 Pre-processing of Points

When the NIR images have been filtered from the structured-light pattern these can be used, together with the depth images, to compute each pixel's 3D point. The same

method as described in section 3.1 is used to find point correspondences between consecutive frames. The interest point detector finds points at corners, which could represent a depth discontinuity. Since the KLT-tracker has sub-pixel accuracy, the values in the depth image must be interpolated in a small neighbourhood. Values on different sides of the discontinuity may differ a lot, and interpolation must in such cases be avoided. We create a validity mask to select which KLT tracks are good. For this, an edge detector is used on each depth image to detect the discontinuities. The thresholded edge map is then expanded by 4 pixels to create the validity mask.

The Kinect's depth map is quite noisy, and this can considerably affect a point's 3D position. In addition to the possibly incorrect z value, the depth values are also multiplied with a point's x and y coordinates after projection through the camera matrix, which gives larger distances between 3D point correspondences, even though image point correspondences in consecutive frames are close. To remove these outliers, point correspondences are fed into a rigid motion estimation RANSAC loop [11]. In each sample, the camera's per frame rotation and translation is estimated using the orthogonal Procrustes method, see e.g. [22]. Even though the 3D point clouds exhibit geometric distortions due to the rolling shutter (which violates the rigid motion model), outliers can still be rejected, by using a high inlier threshold.

4.2 Camera Motion Model

A 3D point in the scene, $\mathbf{X}$, and its projections, $\mathbf{x}$ (homogeneous), in the NIR and depth images, have the following relationship in the camera's coordinate system:

$$\mathbf{x} = \mathbf{K}\mathbf{X} \text{ , and } \quad \mathbf{X} = z(\mathbf{x})\mathbf{K}^{-1}\mathbf{x} \text{ ,} \tag{7}$$

where $\mathbf{K}$ is a 5DOF upper triangular 3×3 *intrinsic camera matrix*, and $z(\mathbf{x})$ is the point's value in the depth image.

We model the camera motion as a sequence of rotation matrices $\mathbf{R}(t) \in \mathrm{SO}(3)$ and translation vectors $\mathbf{d}(t) \in \mathbb{R}^3$. We use the image row N, as time parameter starting at the top row. By calibrating the camera's readout time t_r (see section 5), the inter-frame delay t_d can be calculated and expressed as number of blank rows N_b [9]:

$$N_b = N_r t_d/(1/f) = N_r(1 - t_r f) \,, \tag{8}$$

where f is the frame rate. For an image pair, this gives us the time parameter $N_1 = x_2/x_3$ for a homogeneous point in the first image and $N_2 = x_2/x_3 + N_r + N_b$ for a homogeneous point in the second image, where N_r is the number of image rows.

4.3 3D Motion from Image and Range Video

Since the camera is moving during frame capture, corresponding points in the images will reconstruct different 3D points. Assuming that the 3D points are static, this difference is used to find the camera motion. Assuming a point in the first image is at row N_1, and its corresponding point in the second image is at row N_2, their reconstructed 3D points can be written as $\mathbf{X}_1$ and $\mathbf{X}_2$ for frame 1 and 2, and calculated using the image coordinates and the depth map according to (7). These points can be transformed

to the position $\mathbf{X}_0$, where the reconstructed point should have been, if it was imaged at the same time as the first row in the first image:

$$\mathbf{X_0} = \mathbf{R}(N_1)\mathbf{X}_1 + \mathbf{d}(N_1) \tag{9}$$

$$\mathbf{X_0} = \mathbf{R}(N_2)\mathbf{X}_2 + \mathbf{d}(N_2). \tag{10}$$

The cost function to minimise is thus:

$$J = \sum_{k=1}^{K} ||\mathbf{R}(N_{1,k})\mathbf{X}_{1,k} + \mathbf{d}(N_{1,k}) - \mathbf{R}(N_{2,k})\mathbf{X}_{2,k} - \mathbf{d}(N_{2,k})||^2, \tag{11}$$

where K is the number of point correspondences.

In the experiments we have minimised (11) using the Matlab solver `lsqnonlin` with the *trust-region reflective* algorithm.

We represent each rotation with a three element axis-angle-vector and each 3D translation by another three element vector. This results in 12 unknowns for each point correspondence, which only gives us three equations. In order to solve this, we parametrise rotations and translations with two interpolating splines, using SLERP (Spherical Linear intERPolation) [20] for the rotations and linear interpolation for the translations. Since we fixate the origin at the beginning of a spline, the parameters for this knot do not need to be estimated. The number of spline knots during a frame interval can be chosen depending on how fast the motion is changing and how many correspondences we have. With L number of knots in one of the splines, and M knots in the other, we have $3(L + M - 2)$ unknowns to solve for.

4.4 3D Motion using Known Model

If the 3D structure of the scene is known beforehand, point correspondences (e.g. from SIFT [15]) between the model and the rolling shutter video can be used to find the camera motion. Given 3D point correspondences
$\mathbf{X}_{m,k} \leftrightarrow \mathbf{X}_{1,k}$ we optimise:

$$J = \sum_{k=1}^{K} ||\mathbf{X}_{m,k} - \mathbf{R}(N_{1,k})\mathbf{X}_{1,k} - \mathbf{d}(N_{1,k})||^2, \tag{12}$$

where $\mathbf{X}_m$ is a point in the 3D model and $\mathbf{X}_1$ a point in the point cloud calculated from the Kinect data. Since the position of $\mathbf{X}_m$ is known, we now only have 6 unknowns for each point correspondence. We parametrise the rotations and translations with interpolating splines as before, but now we also have to estimate the rotation and translation for the first knot.

4.5 Rectification

When the camera motion has been estimated, the point clouds can be geometrically rectified to better represent the correct 3D scene. This can be done either per point, if

a sparse number of points is to be rectified, or row-wise, in the dense reconstruction case, since all the pixels within a row share the same transformation. A distorted 3D point $\mathbf{X}_1$ can be transformed to the rectified position $\mathbf{X}'$ by:

$$\mathbf{X}' = \mathbf{R}_{\text{ref}}(\mathbf{R}(N_1)\mathbf{X}_1 + \mathbf{d}(N_1)) + \mathbf{d}_{\text{ref}}, \tag{13}$$

where $\mathbf{R}_{\text{ref}}$ and $\mathbf{d}_{\text{ref}}$ describe a global transformation to a reference coordinate system. By choosing $\mathbf{R}_{\text{ref}} = \mathbf{I}$ and $\mathbf{d}_{\text{ref}} = [0\ 0\ 0]^T$, the point cloud will be transformed to the position the camera had at the first knot in each spline.

Once a 3D point cloud has been rectified, the corresponding depth map and video frame can also be rectified. By projecting the 3D points through the camera matrix, and saving the rectified image coordinates in a new image grid, at the same position as the original points, a forward mapping is created. This is a combination of (7) and (13):

$$\mathbf{x}' = \mathbf{K}[\mathbf{R}_{\text{ref}}(\mathbf{R}(N_1)\mathbf{X}_1 + \mathbf{d}(N_1)) + \mathbf{d}_{\text{ref}}]. \tag{14}$$

The mapping tells us how each pixel (where depth information is available) in the depth map or video frame should be moved to its rectified position $\mathbf{x}'$. Note that since the 3D points are re-projected into the camera, even the noisy depth values can be used here as most of the noise will be removed by the projection. Figure 1 shows an example of a Kinect NIR frame and depth map acquired during fast motion, and the re-projected results from our rectified point cloud.

5 Readout Time Calibration

We have calibrated the rolling shutter readout times for the colour camera (not used in this paper) and the NIR camera in the Kinect, using the method and implementation described in [9]. This involves imaging an LED that flashes with a known frequency, and measuring the width of the resultant stripes in the image. We have used a DSO Nano pocket oscilloscope to generate the square waves that power the LEDs. For the colour camera, we used a plain red LED, and for the NIR camera, we used an Osram SFH 485-2 IR-LED, with 880nm wavelength. For each camera we used six probing frequencies, and thus obtained six readout values. The obtained values are listed in table 2, together with their means and standard deviations.

For the NIR camera, we also noticed a slight drift in the frame delivery. When imaging the IR-LED when flashing at 60 Hz, one should obtain a fixed pattern of horizontal stripes if the camera has a 30 Hz frame-rate. By measuring the slope of these stripes over time we found a correction factor for the frame-rate. Assuming that the frequency delivered by the DSO Nano is correct, the actual frame-rate of the NIR camera is 29.9688 Hz. For the colour camera no such drift was observed.

6 Experiments

We evaluate our algorithm by measuring angles between the visible sides of a cube, before and after rectification. We also compare the closest point distances in the rectified

oscillation (Hz)	readout time (msec)	oscillation (Hz)	readout time (msec)
58	25.7069	57	30.7018
65	26.8769	66	30.4848
87	26.2414	87	30.5977
93	25.914	92	30.5217
115	25.8957	117	30.4957
124	26.0323	122	30.4918
	$\mu = 26.11$ $\sigma_{\text{est}} = 0.17$		$\mu = 30.549$ $\sigma_{\text{est}} = 0.0350$

Table 2: Measured readout times, mean estimate (μ), and std of estimate (σ_{est}). Left: colour camera, Right: NIR camera.

point clouds to the unrectified point clouds. In order to relate the alignment accuracy to rotational velocities, we have computed the sensor rotational velocity using an iPod Touch4 device. Even though rolling shutter artifacts can arise from a fast translation of the sensor (moving platform, such as a car) the dominant cause is usually due to rotation. Our method estimates both translation and rotation, but the sensor has only been rotated in the experiments.

6.1 Ground Truth Rotational Velocities

We have generated several series of panning motions, at various rotational velocities, by rotating the Kinect while mounted on a tripod. During capture of the Kinect data, we also logged actual rotational velocities using an iPod Touch4. The gyro in the iPodTouch4 is surprisingly accurate (we have estimated the standard deviation to be less than 0.7 degrees/second). Synchronization between the devices is obtained by tapping the Kinect, and setting time-shifts to the time where the distortion occurs in each data stream. The setup, and an example gyro log are shown in figure 3.

6.2 Angle estimation from known object

We evaluate our method by comparing the angles between the visible sides of a wooden box, before and after rectification. The three sides of the box are manually segmented in each evaluated frame. Ground-truth angles are obtained by imaging the box when the sensor was stationary, see figure 4. We estimate the plane angles, by first finding the cube normals using RANSAC, with an inlier threshold of 0.01. We then compute the angle between two normals using the formula

$$\Theta_{k,l} = \sin^{-1}(\|\hat{\mathbf{n}}_k \times \hat{\mathbf{n}}_l\|)\,, \tag{15}$$

where $\hat{\mathbf{n}}_k$ and $\hat{\mathbf{n}}_l$ are normal vectors for the two planes.

In figure 5 we plot the estimated angles between the box sides before and after rectification. As can be seen, the angles after rectification are closer to the ground-truth, especially the angles that include the top plane (plane 2).

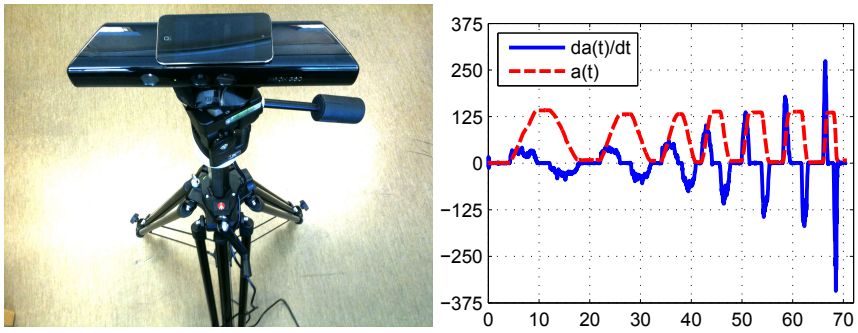


Figure 3: Setup for rotation ground-truth measurement. Left: A Kinect mounted on a tripod, together with an rigidly duct-taped iPod Touch4, with built-in gyro, which are used to estimate rotations. Right: Example of measured rotational velocity in degrees/second (BLUE), and integrated position in degrees (RED).

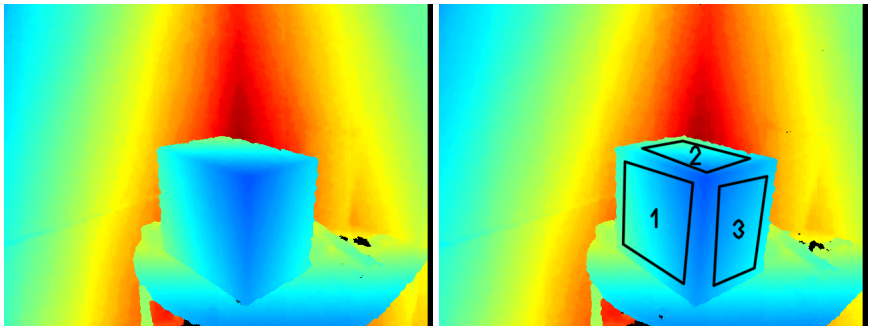


Figure 4: Left: Depth frame from a static sensor. Right: Manually marked planes on frame captured during sensor rotation.

For high rotational velocities (to the right in figure 5), the motion blur is severe, and the plane normals are difficult to estimate accurately. The depth data appears to be less noisy on the top side, which may be the reason why angles that include this side are more accurate. At velocities above about $115°$/sec, depth maps are simply too noisy to be useful. However, the rectified point clouds and depth maps at high velocities still look less geometrically distorted than the originals. See figure 6 for a rectification at a rotational velocity of $172.9°/\text{sec}$.

6.3 ICP Alignment

Another way to evaluate the point cloud rectifications, is to check how well two rectified point clouds can be aligned, compared to the corresponding unrectified point clouds. As evaluation measure we use the distribution of closest point distances after alignment. For two point clouds, we first find the best alignment using the *iterative*

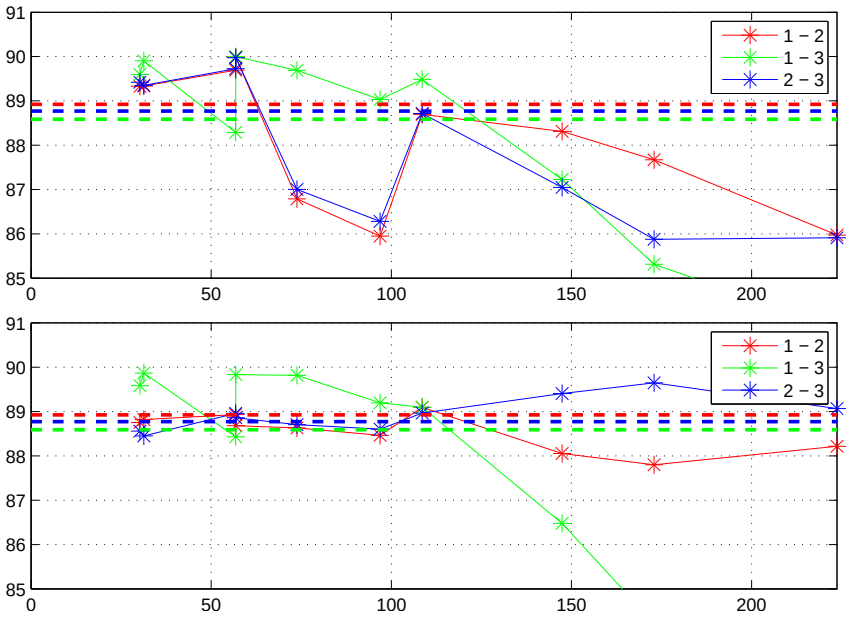


Figure 5: Angles between box sides, as function of rotational velocity. Each colour corresponds to a particular pair of planes, and the horizontal dashed lines show the ground-truth. Top: Box angles before rectification. Bottom: Box angles after rectification. The corresponding plane numbers are given in figure 4.

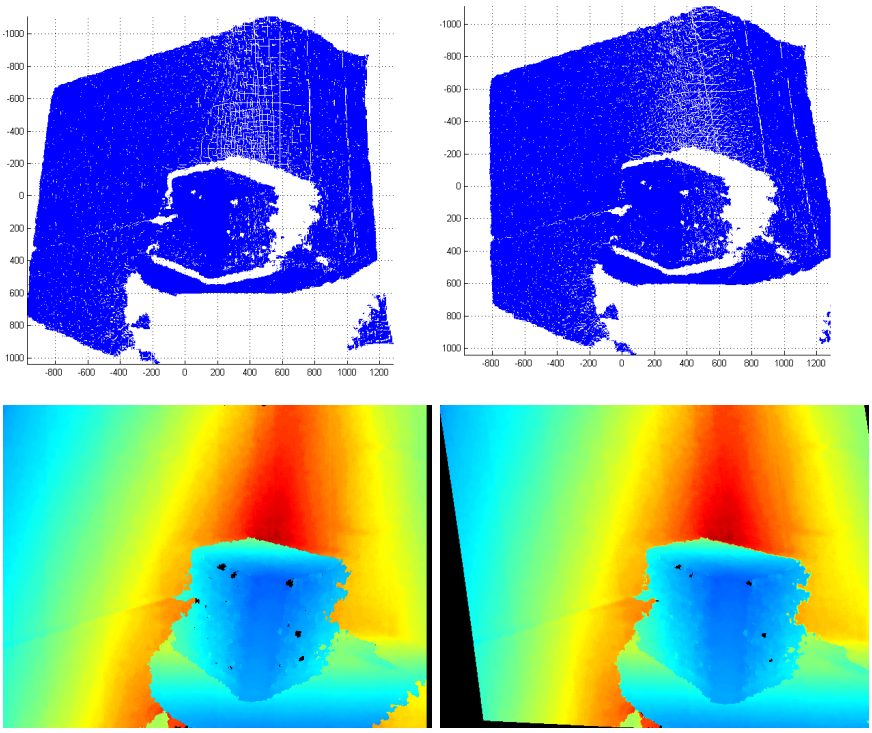


Figure 6: Top row: Unrectified and rectified point clouds at rotational velocity $172.9°/\text{sec}$. Bottom row: Re-projected depth maps

closest point algorithm (ICP) [23]. We then transform one of them using the found transformation, yielding aligned point clouds $\{\mathbf{X}\}_1^M$, and $\{\mathbf{Y}\}_1^N$. Using these, we now compute closest point distances for all points in the first set to the second one:

$$d_m = d(\mathbf{X}_m, \{\mathbf{Y}\}_1^N) = \min_{n \in [1,N]} ||\mathbf{X}_m - \mathbf{Y}_n|| \,. \tag{16}$$

For these distances, we then compute a *kernel density estimate* (KDE) [4], to obtain a probability density curve $p(d)$. Figure 7 shows an example of aligned point clouds, without rectification, and with rectification applied. The corresponding KDE plots are given below. Each two pairs of point clouds have been generated from two images, approximately imaging the same part of the scene, but with different motions. For one of the point clouds in a pair, the sensor was panning left, and for the other one it was panning right. The images have been down-sampled 8 times in each dimension to speed up the ICP calculations. This corresponds to approximately 4800 points per image depending on how much depth data is available.

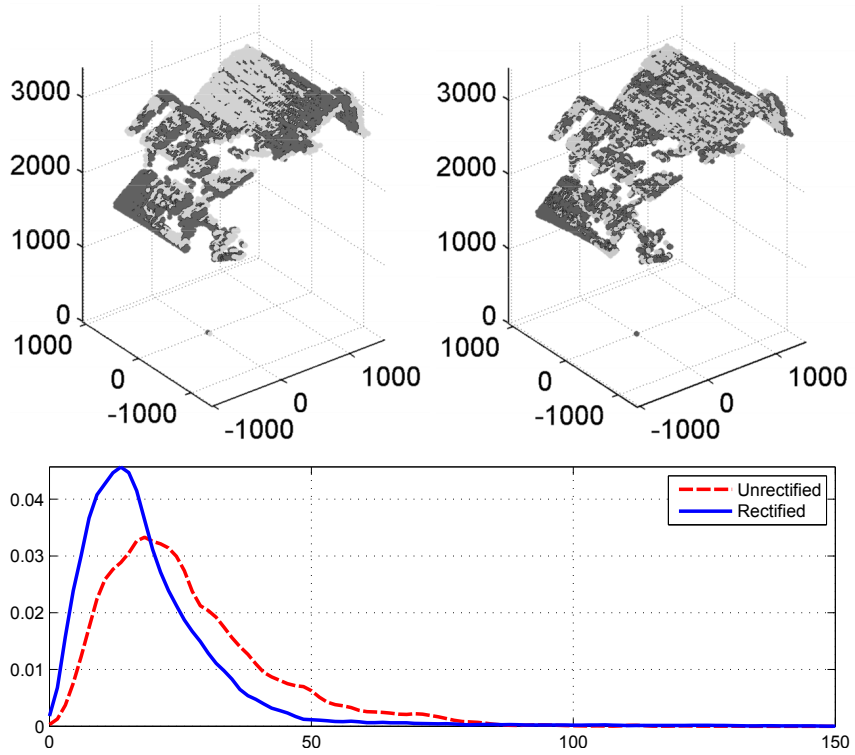


Figure 7: Point cloud alignment at $233.0°/\text{sec}$ relative rotation. Top left: ICP alignment of unrectified point clouds. Top right: ICP alignment of rectified point clouds. Bottom: $p(d)$ for the two alignments.

Δrotation	μ_d(unrectified)	μ_d(rectified)	$\Delta\mu_d$
$71.5°$	24.0907	21.1574	2.9333
$134.3°$	26.1565	20.602	5.5545
$156.8°$	25.7794	20.1405	5.6389
$233.0°$	26.4491	18.994	7.4551

Table 3: Average improvements of alignment errors for various relative rotational speeds. μ_d values are sample means (i.e. the centre-of-gravities for the corresponding KDE curves).

7 Concluding Remarks

We have introduced an algorithm that suppresses rolling shutter distortions caused by device motion and describe how to rectify the 3D point clouds. We also show how the point clouds can be used for rectification of the corresponding depth maps and video frames.

Only one previous attempt to use SLRS with rolling shutters during device motion has been made [12], and it required slow motion of the device. We have demonstrated that our method improves the geometric consistency of the 3D scene by comparing the closest point distances between rectified point clouds with min distances for unrectified ones. We have also shown that our rectification can restore planar surfaces under rotational velocities up to $115°$/sec. At higher velocities the depth maps from the structured light sensor are too noisy to be useful.

A strength and a weakness with the proposed method is that it relies on feature correspondences. While feature correspondences help avoiding the local minima that ICP is sensitive to, fast motions will cause motion blur, which removes many correspondences. Furthermore, if a scene without structure is imaged (e.g. a white wall) no correspondences can be found. For better robustness, more sources of information are needed. For example, one could also use range data features to find correspondences. Another interesting line of future work would be to add a gyroscopic sensor to the structured light device. The proposed rectification scheme could then instead be fed with device motion estimated from the inertial sensors. Such an approach would also work for scenes with neither texture nor 3D structure.

We have found that it is important to have good points as input to the optimisation. This is because with too much noise in the data, the algorithm may not converge. However, in such cases the result is still better than using the Procrustes method or ICP on the unrectified data. We currently use three rejection steps (cross-checking, validity mask, and RANSAC), and in the future we would like to improve and simplify this process. It would also be interesting to feed the estimated camera trajectory as a starting guess to a SLAM, or a bundle adjustment system.

References

[1] Omar Ait-Aider, Nicolas Andreff, Jean Marc Lavest, and Philippe Martinet. Simultaneous object pose and velocity computation using a single view from a rolling shutter camera. In *Proceedings of ECCV'06*, pages 56–68, Graz, Austria, May 2006.

[2] Simon Baker, Eric Bennett, Sing Bing Kang, and Richard Szeliski. Removing rolling shutter wobble. In *CVPR10*, San Francisco, USA, June 2010. IEEE Computer Society, IEEE.

[3] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. In *ICCV07*, Rio de Janeiro, Brazil, 2007.

[4] C. M. Bishop. *Neural Networks for Pattern Recognition.* OU Press, 1995.

[5] Michael Bosse and Robert Zlot. Continuous 3d scan-matching with a spinning 2d laser. In *ICRA09*, Kobe, Japan, May 2009.

[6] Jean-Yves Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker. Technical report, Intel Corporation., 2000.

[7] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: Tutorial part I. *Robotics and Automation Magazine*, 13(2):99–110, 2006.

[8] Jan Elseberg, Dorit Borrmann, Kai Lingemann, and Andreas Nüchter. Non-rigid registration and rectification of 3d laser scans. In *IROS10*, Taipei, Taiwan, October 2010.

[9] Per-Erik Forssén and Erik Ringaby. Rectifying rolling shutter video from hand-held devices. In *CVPR10*, San Francisco, USA, June 2010. IEEE Computer Society.

[10] Christopher Geyer, Marci Meingast, and Shankar Sastry. Geometric models of rolling-shutter cameras. In *6th OmniVis WS*, 2005.

[11] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[12] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *ISER*, December 2010.

[13] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *(ISMAR'09)*, Orlando, October 2009.

[14] Hans Knutsson and Carl-Fredrik Westin. Normalized and differential convolution: Methods for interpolation and filtering of incomplete and uncertain data. In *CVPR93*, pages 515–523, New York City, USA, 1993.

[15] David G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[16] Tuan Q. Pham and Lucas J. van Vliet. Normalized averaging using adaptive applicability functions with applications in image reconstruction from sparsely and randomly sampled data. In *SCIA03*, pages 485–492, 2003.

[17] Primesense reference platform `http://www.primesense.com/files/FMF_2.PDF`.

[18] Stephen Se and Piotr Jasiobedzki. Photo-realistic 3D model reconstruction. In *ICRA06*, Orlando, Florida, May 2006.

[19] Jianbo Shi and Carlo Tomasi. Good features to track. In *CVPR94*, Seattle, June 1994.

[20] Ken Shoemake. Animating rotation with quaternion curves. In *Int. Conf. on CGIT*, pages 245–254, 1985.

[21] Various. Special issues on the 2007 DARPA urban challenge, parts I- III. In Martin Buehler, Karl Iagnemma, and Sanjiv Singh, editors, *JFR*, volume 25. Wiley Blackwell, August 2008.

[22] Thomas Viklands. *Algorithms for the Weighted Orthogonal Procrustes Problem and Other Least Squares Problems*. PhD thesis, Umeå University, 2006.

[23] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *IJCV*, 13(2), 1994.

Chapter D

Co-alignment of Aerial Push-Broom Strips using Trajectory Smoothness Constraints

This is an edited version of the paper:

> Erik Ringaby, Jörgen Ahlberg, Per-Erik Forssén, and Niclas Wadströmer. Co-alignment of aerial push-broom strips using trajectory smoothness constraints. In *Proceedings SSBA'10 Symposium on Image Analysis*, pages 63–66, March 2010.

Co-alignment of Aerial Push-Broom Strips using Trajectory Smoothness Constraints

Erik Ringaby[1], Jörgen Ahlberg[2], Per-Erik Forssén[1], and Niclas Wadströmer[2]

[1]Department of Electrical Engineering, Computer Vision Laboratory
Linköping University, Sweden
[2]Sensor Informatics Group, Swedish Defence Research Agency (FOI),
Linköping, Sweden

Abstract

We study the problem of registering a sequence of scan lines (a *strip*) from an airborne push-broom imager to another sequence partly covering the same area. Such a registration has to compensate for deformations caused by attitude and speed changes in the aircraft. The registration is challenging, as both strips contain such deformations.

Our algorithm estimates the 3D rotation of the camera for each scan line, by parametrising it as a linear spline with a number of knots evenly distributed in one of the strips. The rotations are estimated from correspondences between strips of the same area. Once the rotations are known, they can be compensated for, and each line of pixels can be transformed such that the ground trace of the two strips are registered with respect to each other.

1 Introduction

In airborne remote sensing, push broom imagers are commonly used. Such an imager has a spatial resolution of $1 \times K$ pixels, and exploits the ego motion of the platform on which the imager is mounted to form an image. The imager in itself is thus equivalent to a scanning sensor, but without the scanning mechanics. The platform, typically a fixed-wing aircraft, does not move in a perfectly straight line, and moreover rotates slightly around all three axes. This causes distortions in the resulting image (a sequence of lines), which thus needs registration and orthorectification to be useful for most applications. Hardware for navigation (GPS, INS) and sensor stabilisation can be used to some degree, but are not always available.

Our primary applications are change detection and anomaly detection [1], while mapping and signature-based target detection are secondary. For change detection to be possible at all, the image lines of the two (or more) acquired data strips must be

registered with high precision. In order for any kind of detection (change, anomaly, target) to be georeferenced, the imagery must be registered to other available orthorectified images.

In this paper, we propose a method for mutual registration of push broom data strips. The method is an adaptation of the method developed at CVL for rolling shutter video sensors.

The outline of the paper is as follows. The scenario, the sensor, and the data are described in Section 2, the registration method is described in Section 3, results are shown in Section 4, and conclusions are drawn in Section 5.

2 Sensors and data

The imager, ImSpec, is a visual and near-infrared (391–961 nm) hyperspectral imager from SpecIm [3], with 1024 pixels in each scan line and a maximum of 256 spectral bands. Due to limitations in read-out electronics' data rate, the number of spectral bands might need to be reduced to meet requirements on the number of lines to be acquired per second. In this experiment 60 spectral bands are recorded, which is more than enough for our applications. The imager is mounted nadir-looking in a small fixed wing aircraft as shown in Fig. 1. Data was acquired by flying over approximately the same land strip twice, a rural area at the Swedish Army Ground Combat School premises at Kvarn outside Linköping. The flight altitude was 1000 meters, yielding a pixel footprint of around 0.5 meters. The aircraft was equipped with GPS, but for unknown reasons the GPS data was not logged to the hard disk during the flight. Two resulting data sets are shown in Fig. 2.

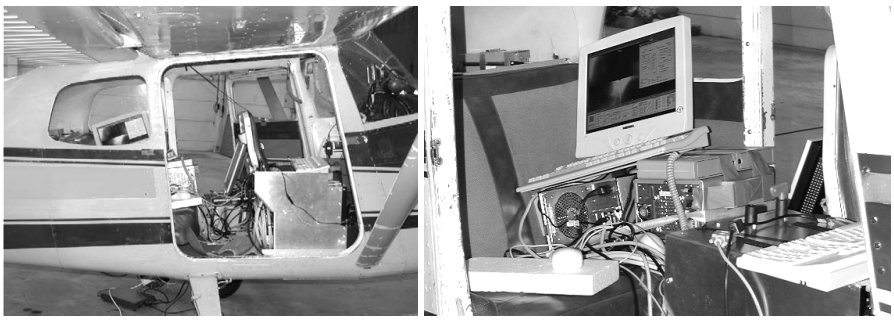


Figure 1: The sensor installation in the aircraft. Left: The sensors are installed in the metal box beside the pilot. Right: Computer and storage equipment in the back seat.

For the purpose of strip alignment, we view each ImSpec strip as one image. For visualisation, and correspondence finding, a mean of three wavelengths approximately corresponding to blue, green and red are used.

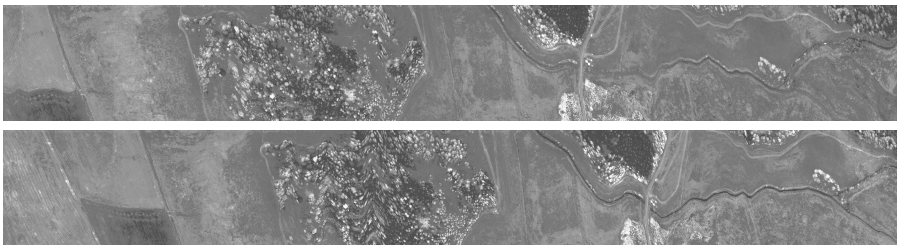


Figure 2: Two strips from approximately the same area.

3 Registration method

Our algorithm estimates the 3D rotation of the camera to compensate for the misregistration due to the aircraft's rotation. The algorithm can be summarised in the following steps:

1. Initial strip alignment using SIFT-features and a global homography model
2. Dense point correspondences with KLT using the initial alignment
3. Separation of the strip into segments where rotation is assumed smooth
4. Rotation estimation
5. Image rectification

3.1 Homography estimation with SIFT-features

Even though the pilot tries to fly over the same area for each strip, the paths may differ (due to e.g. small rotations during the flight). The spatial location of the first row may also differ if the data gathering started at different locations. The aircraft speed may also differ between different strips.

To get a coarse initial alignment of the strips we use SIFT descriptors [5] together with RANSAC to estimate a homography, see Fig. 3. In each strip, SIFT finds between 10 000 and 25 000 features. From these we select the 300 with the highest ratio score, and use these to estimate a homography using RANSAC. We have used an inlier threshold of 50 pixels on the symmetric transfer error [4].

3.2 Point correspondences with KLT

In order to estimate the camera rotation we need point correspondences between the strips. We obtain them by tracking points with the KLT-tracker [6, 7]. The KLT tracker uses a spatial intensity gradient search which minimises the Euclidean distance between the corresponding patches in the different strips. We use the scale pyramid implementation of the algorithm in OpenCV. Correspondences from KLT benefit from the initial homography alignment, and also from first scaling the forward axis of each

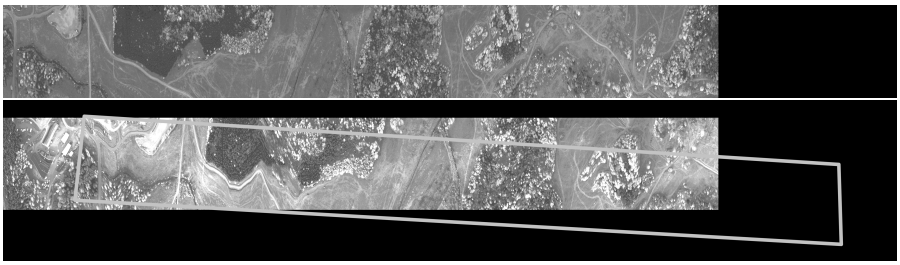


Figure 3: Initial homography alignment found with SIFT and RANSAC. Top: strip 1, bottom: strip 2, with transformed bounding box from strip 1 overlaid.

dataset by a factor of two. The first image is initialised with a regular grid of points which are tracked and re-tracked. This means that when a point is tracked from the first strip to the other one, it is tracked again and if it returns to the original position the point is regarded as a match [2]. This procedure removes outliers efficiently.

3.3 Separation of the strip into segments

The rotations during a flight are assumed to vary smoothly, and thus the strip can be split into several segments. At each line that splits two segments, we introduce a *key-rotation*, i.e. a knot in a linear spline that interpolates aircraft rotations. The number of key-rotations needed may differ depending on how long the strip is, and how big the aircraft rotations were (i.e. if the strips differ much). We have chosen to have N key-rotations uniformly spaced in the first strip and to use local linear regression to calculate where these rotations approximately should be in the other strips, see Fig. 4.

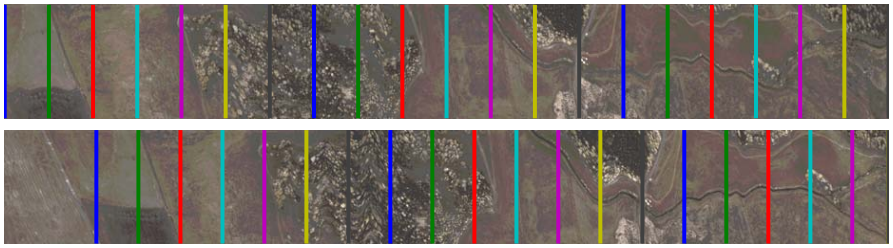


Figure 4: Rotation distribution with N = 21

3.4 Rotation estimation

We model the distortion of a strip as a sequence of rotation homographies:

$$\mathbf{H}(t) = \mathbf{K}\mathbf{R}(t)\mathbf{K}^{-1} \,, \tag{1}$$

i.e. we neglect the translational component of the aircraft motion. This means that we model the sensor as rotating purely about its optical centre, and thus the imaged ground patch is modelled as being on the interior surface of a sphere. This is bound to cause some distortions in the reconstruction, but if the radius of the sphere (i.e. the focal length in $\mathbf{K}$) is large enough (compared to the strip length), this distortion is small.

We optimise for a sequence of rotations $\mathbf{n}_1,\ldots,\mathbf{n}_N$ using the MATLAB optimiser `lsqnonlin`. The optimisation makes use of a cost function

$$J = \epsilon(\mathbf{n}_1,\ldots,\mathbf{n}_N)\,, \tag{2}$$

with image correspondences $\mathbf{x}_k \leftrightarrow \mathbf{y}_k$ as constant parameters. The rotations $\mathbf{n}_1,\ldots,\mathbf{n}_N$ are represented as three element vectors, where the magnitude corresponds to the rotation angle, and the direction is the axis of rotation, i.e. $\mathbf{n} = \varphi\hat{\mathbf{n}}$. This is a minimal parametrisation of rotations, and it also ensures smooth variations, in contrast to e.g. Euler angles. The vector $\mathbf{n}$ can be converted to a rotation matrix using the matrix exponent. Because we are dealing with only rotations this can be simplified to Rodrigues formula.

In order to make the optimisation more stable, the first and last key-rotations in the first strip is set to identity rotations. To help the global optimisation with an initial guess, smaller segments (areas between vertical lines in figure 4) is locally optimised and used as input to the global optimisation.

We have also augmented the cost function with a regularisation, where we put an extra cost on large changes between consecutive rotations. This trajectory smoothness constraint is done by adding a term in the cost function:

$$J = \epsilon(\mathbf{n}_1,\ldots,\mathbf{n}_N) + \alpha\left(\sum_{l=1}^{N-1} 1 - \hat{\mathbf{n}}_l^T\hat{\mathbf{n}}_{l+1}\right)\,. \tag{3}$$

This regularisation is necessary, otherwise the optimiser may find very strange trajectories, see Fig. 6.

3.5 Image rectification

When the key-rotations have been estimated, they can be used to assign a rotation to each row using SLERP (Spherical Linear intERPolation) [8]. With this we know how each point should be displaced in order to rectify the scene.

We have chosen to perform the rectifying interpolation in three steps: First, we create an all-zero RGBA image. Second, we apply the rectification to each pixel in the strip image. The 3×3 closest grid locations are then updated by adding vectors of the form (wr, wg, wb, w). Here r, g, b are the colour channel values of the input pixel, and w is a variable weight that depends on the grid location $\mathbf{y}$, according to:

$$w(\mathbf{y}) = \exp(-.5(\mathbf{y}-\mathbf{x}')^2/\sigma^2)\,. \tag{4}$$

Here $\mathbf{x}'$ is the sub-pixel location of the pixel, and σ is a smoothing parameter, which we set to $\sigma = 0.15$. Third, after looping through all pixels, we convert the RGBA image to RGB, by dividing the RGB values by the fourth element. The whole operation is quite

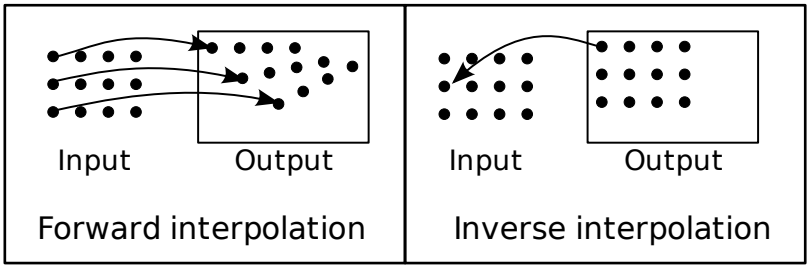


Figure 5: Forward and inverse interpolation. Left: In forward interpolation, we rectify the locations of all pixels to obtain an irregular grid. Neighbours are now correctly defined by the irregular grid. Right: In regular interpolation, one uses the inverse mapping to find out where to sample points. Neighbours are then defined by the regular grid where the distorted pixels lie.

fast, and its parallel nature makes it well suited to a GPU implementation. The number of channels may be changed to correspond to the correct number of bands.

Alternatively, the irregular grid of pixels can be resampled to a regular grid, by defining a triangular mesh over the points, and sampling the mesh using bi-cubic interpolation. This is done by the function `griddata` in Matlab. This method gives a good result but is much slower than the previous proposed method.

Finally, it is also tempting to use regular, or *inverse interpolation*. We can now loop over all values of output pixels, and use inverse mapping to find the pixel locations in the distorted image, and cubically interpolate these, see Fig. 5, right. This is fast but it does not give as good a result as the previously proposed methods because the interpolation takes place in the distorted image.

4 Results

The results presented here are from one of the three strips acquired during the flight. Fig. 6 shows the result when the regularisation parameter is set to zero. The strips are overlapping but it is difficult to make use of the images. When using $\alpha = 60\,000$ as regularisation parameter the trajectory is much smoother and the images look more like the original scene, see figure 7.

5 Conclusion and future work

In order to exploit the hyper-spectral imagery for the mentioned targeted applications, there are additional steps that need to be taken. Where the strips are overlapping in figure 8 left, many false change detections are made. The pixel registration needs to be more exact, which can be achieved by exploiting navigation data (GPS, INS) from the aircraft as well as a suitable motion model. Also additional imagery can be used for

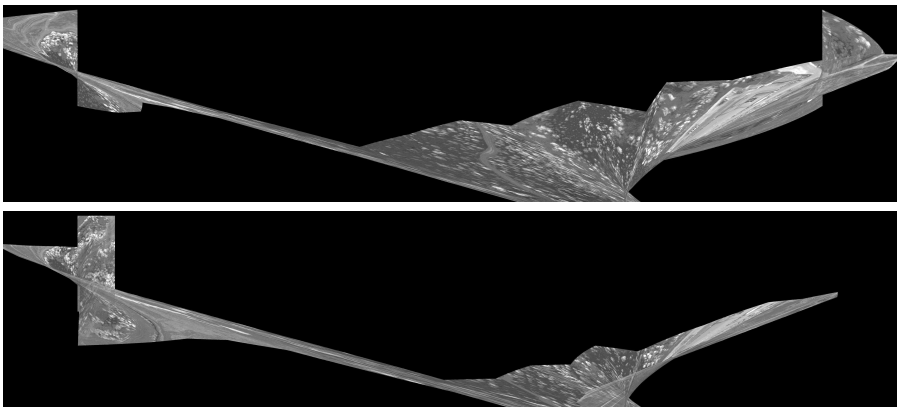


Figure 6: Result for aligning the two strips from the first flight path, without regularisation of large deviations in rotation.

registration. For registration, simultaneously acquired imagery from a staring sensor with high spatial and low spectral resolution (for example a consumer digital camera) can be used, possibly enabling registration exact enough for spectral change detection.

In general, we got better results the more rotations we added. The downside was much higher computation time, and additionally the optimisation had an increasingly harder time to converge.

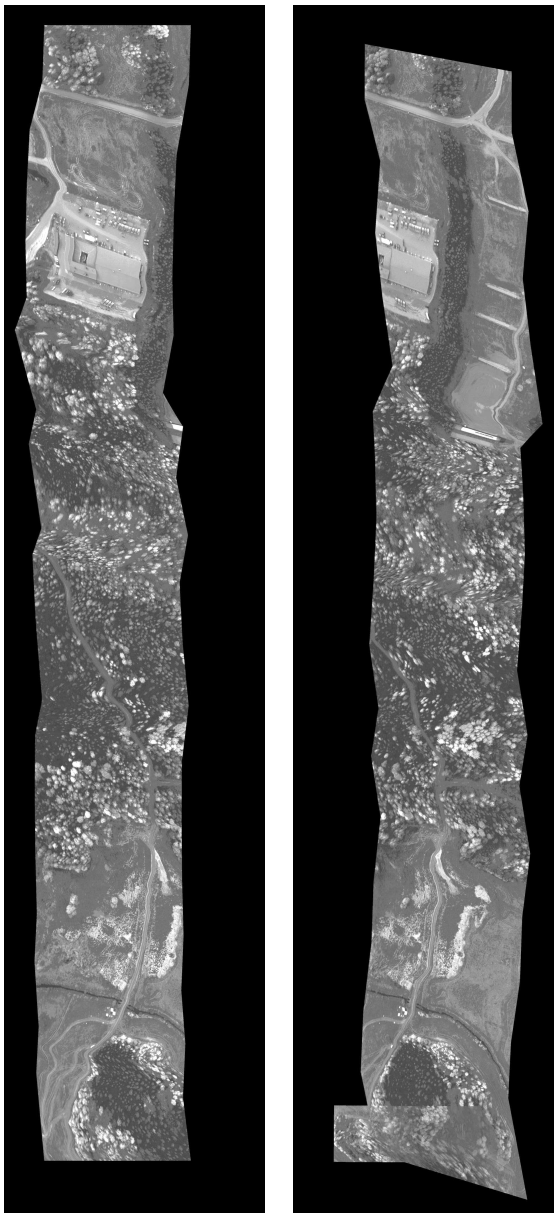


Figure 7: Result for aligning the two strips from the first flight path, with $N = 21$, and regularisation parameter $\alpha = 60\,000$.

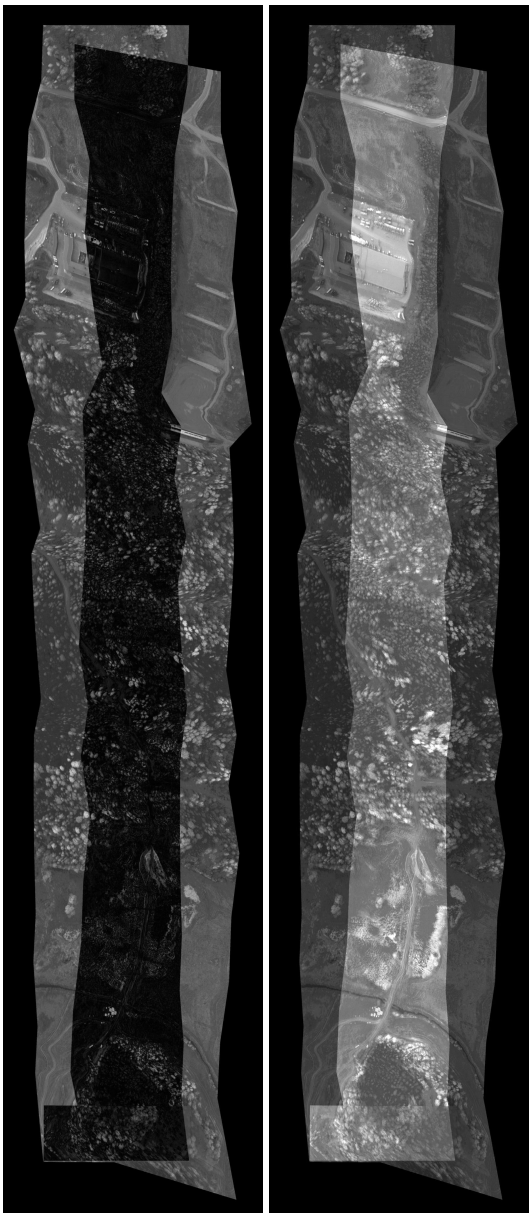


Figure 8: Left: The difference image for the two strips in figure 7.
Right: The average of the two images.

Acknowledgement

The authors would like to thank Dr. Thomas Svensson at the IR Systems Group at FOI for the data. The flights were performed by SkyMovies AB.

References

[1] J. Ahlberg and I. Renhorn. Multi- and hyperspectral target and anomaly detection. Technical Report FOI-R--1526--SE, Swedish Defence Research Agency (FOI), December 2004.

[2] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. In *IEEE ICCV*, 2007.

[3] T. Chevalier. Samverkande sensorer. illustrerat exempel med hyperspektral kamera och 3d-laserradar. Technical Report FOI-R--2325--SE, Swedish Defence Research Agency (FOI), September 2007.

[4] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[5] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, January 2004.

[6] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI81*, pages 674–679, 1981.

[7] Jianbo Shi and Carlo Tomasi. Good features to track. In *CVPR94*, Seattle, June 1994.

[8] Ken Shoemake. Animating rotation with quaternion curves. In *Int. Conf. on CGIT*, pages 245–254, 1985.

Chapter E

Anisotropic Scattered Data Interpolation for Pushbroom Image Rectification

This is an edited version of the paper:

> Erik Ringaby, Ola Friman, Per-Erik Forssén, Thomas Opsahl, Trym Vegard Haavardsholm, and Ingebjørg Kåsen. Anisotropic scattered data interpolation for pushbroom image rectification. *IEEE Transactions in Image Processing*, 2014.

Digital Object Identifier: 10.1109/TIP.2014.2316377

Anisotropic Scattered Data Interpolation for Pushbroom Image Rectification

Erik Ringaby[1], Ola Friman[2], Per-Erik Forssén[1], Thomas Opsahl[3], Trym Vegard Haavardsholm[3], and Ingebjørg Kåsen[3]

[1]Department of Electrical Engineering, Computer Vision Laboratory
Linköping University, Sweden
[2]Swedish Defence Research Agency (FOI), Linköping, Sweden
[3]Norwegian Defence Research Establishment (FFI), Kjeller, Norway

Abstract

This article deals with fast and accurate visualization of pushbroom image data from airborne and spaceborne platforms. A pushbroom sensor acquires images in a line-scanning fashion, and this results in scattered input data that needs to be resampled onto a uniform grid for geometrically correct visualization. To this end, we model the anisotropic spatial dependence structure caused by the acquisition process. Several methods for scattered data interpolation are then adapted to handle the induced anisotropic metric and compared for the pushbroom image rectification problem. A trick that exploits the semi-ordered line structure of pushbroom data to improve the computational complexity several orders of magnitude is also presented.

1 Introduction

Pushbroom scanners are common in multi- and hyperspectral imaging applications from satellite and airborne platforms. A pushbroom scanner has a linear array of sensor elements oriented perpendicular to the flight direction, see Fig. 1. Image data is acquired in a line-by-line fashion as the carrier platform moves forward and the footprint of each scan line therefore depends on the position, velocity, and orientation of the platform at the time of acquisition. Since the platform motion and the topography of the scene typically prevent neighboring scan lines from being parallel, a direct stacking of scan-lines generates a geometrically distorted image, see Fig. 2 top. To produce a geometrically correct *rectified* image, as shown in Fig. 2 bottom, two problems must be solved: (1) *georeferencing* and (2) *scattered data interpolation*. A georeferencing algorithm assigns a world coordinate to each acquired data point, e.g., in the standard WGS84 system [10, 16, 19], by projecting the point down to the ground. This can be done based on a camera model and navigation data from an on-board INS system [3].

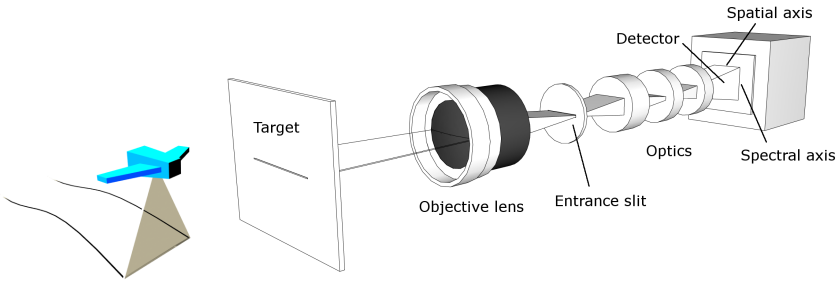


Figure 1: Left: A pushbroom camera acquires images by scanning the ground surface in a line-wise fashion. Right: A hyperspectral pushbroom camera splits the light that enters through the entrance slit into different spectral bands using a dispersive element such as a prism.

Accurate georereferencing also requires a Digital Surface Model (DSM) to determine where the back-projection of a data point intersects the ground surface. The georeferenced pushbroom data constitutes a set of scattered or irregularly sampled points on the ground surface (i.e. each point has a longitude, a latitude, and an elevation). By interpolating this scattered data onto a uniform longitude-latitude sampling grid, a geometrically correct image is produced, see Fig. 2 bottom.

The focus of this paper is on the scattered data interpolation problem. The general problem of scattered data interpolation has received much attention in the literature and several surveys are available [9, 14, 1]. Due to the varying sample density, scattered data interpolation is a much more challenging problem than interpolation of data in a uniform grid. There is also a computational aspect that is related to neighbor relations: in a uniform grid the nearest sample neighbors to an arbitrary point are quickly identified, whereas the distance to *all* sample points must be calculated in an irregularly sampled data set to find the closest sample neighbors to an arbitrary point. This difference leads to two different schemes of scattered data interpolation considered in this work: forward and inverse interpolation, see e.g. [23]. In a forward interpolation scheme, each sample point in the irregular input data is spread onto the neighbor locations in the uniform output grid. This is a fast operation as the sample neighbors are trivially located in the uniform output grid. The main disadvantage with forward mapping schemes is that it is difficult to guarantee that each point in the output grid receives any contribution, i.e., there is a risk that holes with undefined values are obtained in the interpolated image. Therefore, in image interpolation and resampling in general, inverse mapping schemes are preferred to ensure that each position in the output grid is assigned a value. An inverse mapping means that each point in the output grid is mapped back to the input domain where the interpolation takes place by a weighted sum of the neighboring input samples. However, when the input data is irregularly sampled, one is faced with the computational problem of identifying the neighbors, as discussed above. As pushbroom data sets typically are large, this computational aspect becomes an issue to consider.

This work evaluates a number of different techniques for interpolating pushbroom

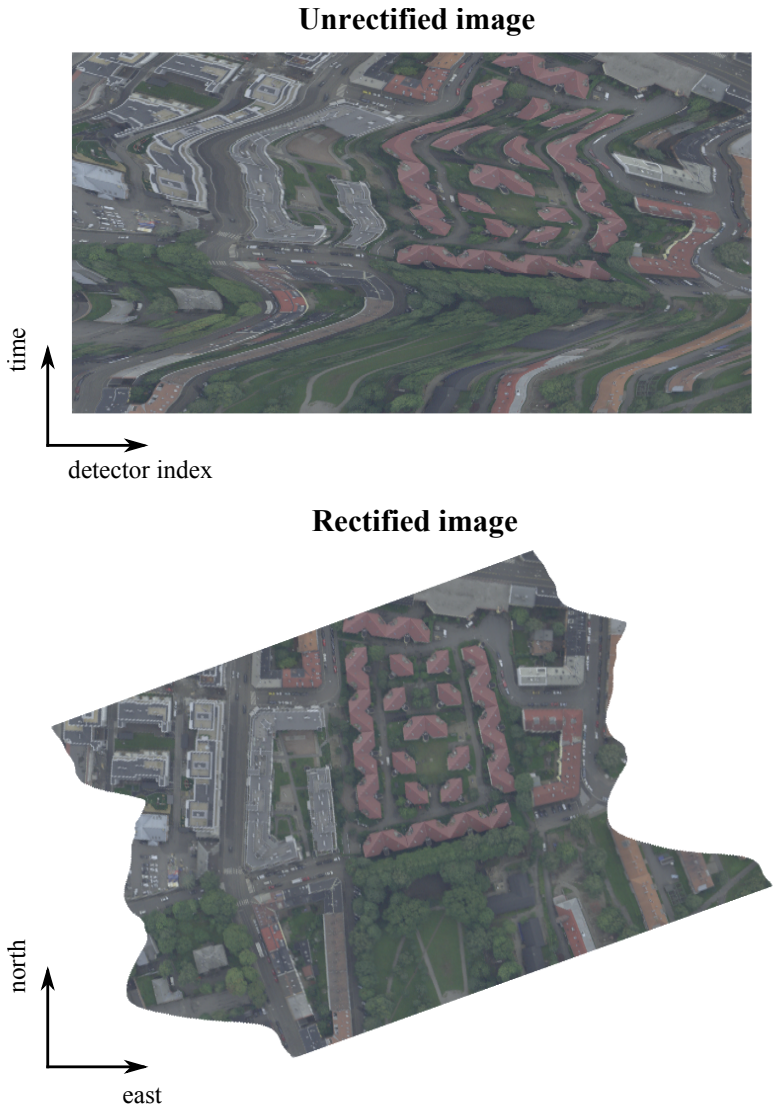


Figure 2: Top: Example of an unrectified pushbroom swath over the city of Oslo. The image consists of stacked lines acquired over time. Bottom: Rectified version of the image in a world coordinate system.

data to produce a rectified image. In the forward interpolation category, the Splatting method is evaluated [26]. This method performs the forward spreading of values using a radial basis function such as a Gaussian kernel. This technique has, for example, previously been used in correction of rolling shutter video [8], which utilizes an acquisition technique similar to the pushbroom acquisition. In the inverse interpolation category, Nearest Neighbor (NN), Inverse Distance Weighted (IDW), Kriging and triangulation-based interpolation methods are included in the comparison. NN interpolation simply uses the closest input sample as the interpolated value. IDW, also known as Shepard's method [21], calculates a weighted sum of the neighboring input samples with the weights equal to the inverse distances from the input samples to the point to interpolate. The Kriging interpolation similarly finds the weights by minimizing a reconstruction error in a least-squares sense, assuming knowledge of the spatial covariance function governing the statistical dependence between samples [13, 15], which typically decays monotonically with distance. Finally, in triangulation-based techniques, neighbor relations in the scattered input data are first established in a pre-processing step, e.g., using a Delaunay triangulation [1]. For triangulated data, interpolation can be made very efficient as the neighbor relations are given, but for large input data sets, the triangulation itself can be expensive. The triangulation-based technique evaluated here is the Natural Neighbor (NAT) method [5, 1]. In contrast to the previous inverse interpolation schemes, which weight input samples according to the distance to the point to interpolate, NAT uses an area-based measure to compute the weights. Other possible interpolation methods include thin-plate splines [4] and Non-Uniform-Rational-B-Spline (NURBS) surfaces [18]. These methods are typically used for smooth approximations in computer graphics and Computer Aided Design applications and require large linear equation systems to be solved, so they are not the first choice for the pushbroom rectification problem. Comparisons of methods for scattered data interpolation have been performed for different applications other than the pushbroom imaging one, see [6, 7] and references therein, with mixed conclusions as to the best method to apply. To the best of the authors' knowledge, an evaluation for the specific pushbroom imaging application has not been presented previously.

There are a number of contributions of the present work. First, the spatial dependence structure of pushbroom data is modeled and shown to be inherently anisotropic, i.e., data correlation is different in different directions. Second, five methods for scattered data interpolation are extended to handle the anisotropic nature of pushbroom data and compared for the image rectification problem. Third, a method that utilizes the semi-structured sampling pattern of a pushbroom sensor to significantly speed up inverse interpolation schemes is presented.

2 Problem description

The problem we consider in this work is to resample pushbroom data onto a uniform grid to obtain a geometrically correct rectified image. We assume that the pushbroom data has been georeferenced so that each pushbroom sample has an associated world coordinate in a coordinate frame for the Earth, e.g., the standard WGS84 system [10, 16, 19]. For a nadir looking sensor, the pushbroom samples are structured in an

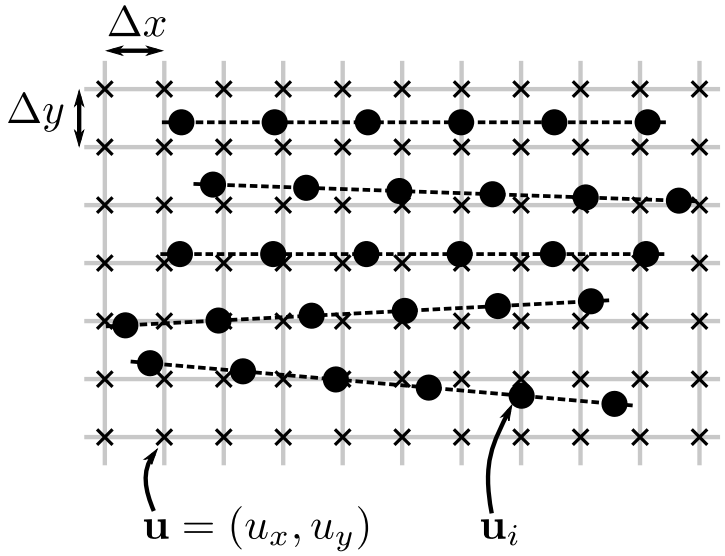


Figure 3: Illustration of the scattered data interpolation problem. Irregularly spaced samples $\mathbf{u}_i$ (black dots) are located on pushbroom lines (dashed). The rectified image is produced by resampling onto the points in a regular output grid (crosses).

approximate line pattern, see Fig. 3. However, both the distance between points on a line and the distance between lines are non-uniform due to, among other factors, perspective geometry and variations in carrier platform velocity and attitude. The set of pushbroom points is therefore irregular. We denote a 2D position in world coordinates by $\mathbf{u} = (u_x, u_y)$ and the specific positions of the pushbroom samples are enumerated by a subscript $\mathbf{u}_i$. For each pushbroom point $\mathbf{u}_i$ there is a measured radiance value $z(\mathbf{u}_i)$. Multi- and hyperspectral pushbroom sensors generate multi-valued vectors with radiance values for different wavelength bands at each sample point. While both the spectral and spatial dimensions can be considered when interpolating new sample values, we focus on the spatial dimension in this work. The rectification is then performed for one wavelength band at a time.

The uniform output grid is user-defined. Typically, one defines the grid as the bounding box of all, or of a subset, of the pushbroom lines, and with a spatial spacing Δx and Δy of the same magnitude as the spacing between the input samples. The rectified image is produced by interpolating values at the uniform grid points. We denote by $\hat{z}(\mathbf{u})$ the interpolated value at an arbitrary position $\mathbf{u}$.

3 Modeling

Successful data interpolation relies on a certain degree of data correlation that can be exploited to predict data values at arbitrary locations. This section investigates and models the dependence structure of pushbroom data in terms of the spatial autocovari-

ance function

$$\rho(\Delta\mathbf{u}) = \text{cov}(z(\mathbf{u}_i), z(\mathbf{u}_i - \Delta\mathbf{u})) \tag{1}$$

that describes the covariance between a pushbroom sample and a point at a distance $\Delta\mathbf{u}$. It is shown that the covariance function is both non-stationary and anisotropic, a fact that must be considered when interpolating new sample values in the rectified uniform grid. As we are working with georeferenced input coordinates, the unit of $\Delta\mathbf{u}$ is meters. For pushbroom and remote sensing images in general, covariance is induced by two main components:

1. Measurement covariance arises when sensor elements integrate light from the same surface region. A *pixel footprint function* (PFF) describes the region over which a sensor element integrates light.

2. Structural autocovariance of the ground surface reflectance $z_r(\mathbf{u})$ that one ultimately is interested in imaging.

While it is clear that the structural autocovariance generally is both non-stationary and locally anisotropic depending on the ground surface structure, it is shown below that the same also holds for the measurement covariance for pushbroom data. For multispectral imaging sensors, there may also be a non-negligible covariance along the spectral dimension that can be exploited to improve the interpolation. If the spectral sensitivity curves, i.e., the spectral footprints, of each band are known, the overlap between these defines a covariance in the measured data across spectral bands. In this work, however, the bands are treated as independent and interpolated separately.

In the sections below, we first derive a model of the pixel footprint function. We then introduce the surface structure model and finally combine these in a measurement model to obtain the total spatial covariance function $\rho(\Delta\mathbf{u})$ in (1).

3.1 Pixel footprint function

Each detector element in the pushbroom sensor integrates reflected light from a surface region described by a Pixel Footprint Function (PFF) $f(\mathbf{u})$. The PFF describes the relative contribution from each point on the ground surface to the measured sample value and it therefore integrates to 1. Note that the PFF is the reciprocal of the Point Spread Function (PSF), $p(\mathbf{x})$, that describes how light from a point source is imaged onto different sensor elements at positions determined by $\mathbf{x}$ in the image plane. Specifically, the PFF is the reprojection of the PSF onto the ground plane, assuming that the topography is locally flat. It is assumed that the georeferenced coordinate of each input data sample is located in the middle of its PFF. Input samples with overlapping PFFs receive light from the same ground area and will therefore be correlated.

For pushbroom sensors, the PFF is determined by an optical component and a motion component. The optical component is obtained directly from the sensor specification in terms of the so-called Instantaneous Field of View (IFOV), which is the angle subtended by each detector element in the pushbroom sensor. This angle defines a ground-projected pixel footprint at any given time. The IFOV may be different in the along-track and across-track directions of the carrier platform, i.e., pixel footprints are

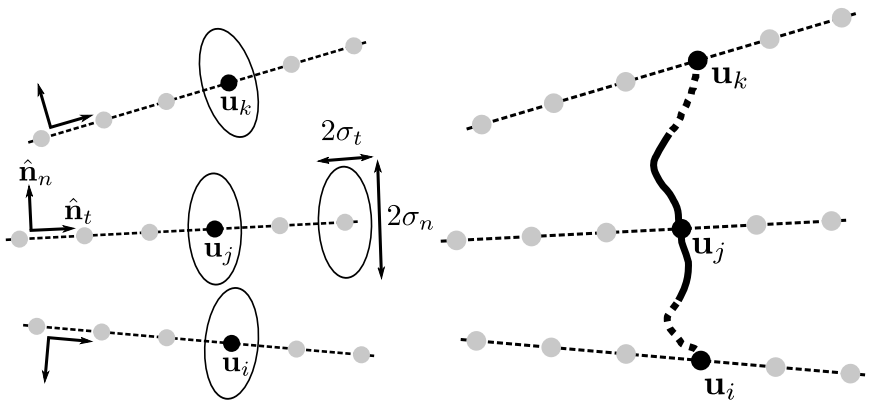


Figure 4: Components in the pixel footprint function. Left: The optical component, $f_{IFOV}(\mathbf{u})$ is an anisotropic Gaussian function, here illustrated with an elliptic contour line, with size σ_n normal to the pushbroom line, and σ_t tangent to the line. Right: The motion component $f_M(\mathbf{u})$ is an integration over the ground plane trajectory between input sample locations in the output grid. The solid part of the curve corresponds to the integration of the sample at $\mathbf{u}_j$.

generally anisotropic. For example, the pushbroom sensor used in this work has an along-track IFOV angle that is twice that of the across-track angle. Although the IFOV typically is specified as a rectangle in angular space, it is modified by both the entrance slit and the lens of the sensor optics, see Fig. 1, right, which both cause diffraction. The diffraction pattern is described by the so-called Airy function, which is often approximated by a Gaussian function when describing the PSF [12]. The shape of the PFF defined by the IFOV can therefore also be modeled by a Gaussian function:

$$f_{IFOV}(\mathbf{u}) \propto e^{-\frac{1}{2}\mathbf{u}^T\mathbf{F}_{IFOV}^{-1}\mathbf{u}}, \tag{2}$$

where $\propto$ means that the scaling factor required for $f_{IFOV}(\mathbf{u})$ to integrate to 1 is omitted. For brevity and without loss of generality, we also assume that the point sample we consider is centered at the origin of the coordinate system. The potentially anisotropic form of the PFF is modeled by defining the metric tensor in (2) as

$$\mathbf{F}_{IFOV} = \sigma_t^2\hat{\mathbf{n}}_t\hat{\mathbf{n}}_t^T + \sigma_n^2\hat{\mathbf{n}}_n\hat{\mathbf{n}}_n^T, \tag{3}$$

with different extents σ_t and σ_n in the tangential and normal directions $\hat{\mathbf{n}}_t$ and $\hat{\mathbf{n}}_n$ of the pushbroom line respectively, see Fig. 4, left. The parameters σ_t and σ_n have the unit meters and they can be derived from the IFOV specification of the pushbroom sensor and the distance to the ground.

The second component affecting the pushbroom PFF is motion blur. This is caused by the integration time during which the shutter remains open while the carrier platform moves forward, and can be thought of as a function $f_M(\mathbf{u})$ that is non-zero along the scan trajectory of a particular sensor element, and that integrates to 1, see Fig. 4, right.

A zeroth order approximation of $f_M(\mathbf{u})$ is a straight line of constant length l in the normal direction $\hat{\mathbf{n}}_n$ for all samples $\mathbf{u}_i$. To obtain a tractable expression for the total PFF, we further approximate this line with a degenerate Gaussian-shaped function

$$f_M(\mathbf{u}) \propto e^{-\frac{1}{2}\mathbf{u}^T \mathbf{F}_M^{-1} \mathbf{u}} \tag{4}$$

with a metric tensor defined to have no extent in the tangent direction so that it is only non-zero along a line

$$\mathbf{F}_M = 0\, \hat{\mathbf{n}}_t \hat{\mathbf{n}}_t^T + \sigma_l^2 \hat{\mathbf{n}}_n \hat{\mathbf{n}}_n^T. \tag{5}$$

The parameter σ_l is set so that the Gaussian function approximates a box function of length l, e.g., $\sigma_l = l/2$. The motion model can be extended to account for curved trajectories using a higher order approximation to account for sideway motions caused by turbulence. The metric tensor in (5) would then have some extent in the tangent direction too and still be applicable in the continued derivation below.

The total pushbroom PFF is now obtained as the IFOV PFF convolved with the motion PFF, using the fact that the convolution of two Gaussians yields another Gaussian:

$$f(\mathbf{u}) = \left(f_{IFOV} * f_M\right)(\mathbf{u}) \propto e^{-\frac{1}{2}\mathbf{u}^T \mathbf{F}^{-1} \mathbf{u}} \tag{6}$$

with

$$\mathbf{F} = \mathbf{F}_{IFOV} + \mathbf{F}_M = \sigma_t^2 \hat{\mathbf{n}}_t \hat{\mathbf{n}}_t^T + \left(\sigma_n^2 + \sigma_l^2\right) \hat{\mathbf{n}}_n \hat{\mathbf{n}}_n^T. \tag{7}$$

Hence, the forward motion of the carrier platform effectively stretches the footprint induced by the optics.

3.2 Surface structure model

The structure of the surface reflectance can be characterized by its autocovariance function $z_r(\mathbf{u})$. For homogeneous regions the function decays slowly and for rugged surfaces it has a faster decay. Along structures such as roads or roof edges, the covariance decays slowly along the structures and quickly across. Hence, the structural autocovariance in an image is generally both non-stationary and anisotropic. A Gaussian model of an anisotropic surface covariance structure is

$$\rho_s(\Delta\mathbf{u}) = \sigma_s^2\, e^{\Delta\mathbf{u}^T \mathbf{S}^{-1} \Delta\mathbf{u}}. \tag{8}$$

In this model, σ_s^2 represents the general magnitude of the variations on the ground surface. The potentially anisotropic covariance matrix $\mathbf{S}$ controls the smoothness of the surface and the autocovariance decays with spatial distance in a Gaussian-shaped fashion. Adaptive image processing approaches that try to adapt to the local structural autocovariance, for example edge-preserving filtering that filters along edges but not across them, have been suggested, e.g., steerable filters [2] and anisotropic diffusion [25]. These ideas were adapted to irregularly sampled data in [24] using alternating optimization of smoothing parameters and local structure.

In this work, a variant of the surface structure modeling method described in [24] is used. First, a structure tensor is estimated in each output pixel, $\mathbf{u}$, as the weighted

outer product of gradients:

$$\mathbf{T} = \sum_{\mathbf{u}_k \in \mathcal{N}(\mathbf{u})} g(\mathbf{u}_k, \sigma) \nabla z(\mathbf{u}_k) \nabla z(\mathbf{u}_k)^T . \tag{9}$$

The neighborhood $\mathcal{N}(\mathbf{u})$ is a set of 7×7 neighbors determined in the input grid, and $g(\mathbf{u}, \sigma)$ is a Gaussian decay. Let $\mathbf{e}_1$, $\mathbf{e}_2$ be the eigenvectors and $\lambda_1 > \lambda_2 > 0$ the eigenvalues of $\mathbf{T}$ respectively. A surface structure covariance matrix is then constructed as

$$\mathbf{S} = \sigma_i^2 \, \phi(\lambda_2) \left[\mathbf{I} - \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2} \mathbf{e}_1 \mathbf{e}_1^T \right], \tag{10}$$

where σ_i^2 is a global scaling factor and

$$\phi(\lambda) = \begin{cases} 1 - \lambda/\lambda_{max} & \text{if } \lambda \leq \lambda_{max}, \\ 0 & \text{if } \lambda > \lambda_{max}. \end{cases} \tag{11}$$

The function $0 \leq \phi(\lambda) \leq 1$ is close to 0 when there is much surface structure, as indicated by a large λ_2, leading to a low overall covariance. If λ_2 instead is small there is at least one direction in which the covariance should be large. The parameter λ_{max} indicates the edge strength in the image above which the covariance should be zero. For the data in this work, which is restricted to the range [0,1], $\lambda_{max} = 0.05$ was a good value. The factor $\frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2}$ is close to 0 for the isotropic surface structure case $\lambda_1 \approx \lambda_2$, leading to an isotropic $\mathbf{S}$, but at edges where $\lambda_1 >> \lambda_2$ this factor is close to 1, indicating a strong anisotropic structure in the $\mathbf{e}_1$ direction along which the covariance should be low. For reference, an isotropic surface structure covariance is also considered in the experiments

$$\mathbf{S} = \sigma_i^2 \mathbf{I}. \tag{12}$$

3.3 Total pushbroom sample covariance

Using the notation introduced above, i.e., $f(\mathbf{u})$ for the Pixel Footprint Function and $z_r(\mathbf{u})$ for the true surface image, a pushbroom data sample is generated from the light integrated by a sensor element plus a noise term:

$$z = \int_{\mathbb{R}^2} f(\mathbf{u}) z_r(\mathbf{u}) \, d\mathbf{u} + \epsilon. \tag{13}$$

The goal is to find the autocovariance function $\rho(\Delta\mathbf{u})$ in (1) of such a sample. By inserting (13) into (1) and simplifying the following expression is obtained

$$\rho(\Delta\mathbf{u}) = C \, e^{-\Delta\mathbf{u}^T (\mathbf{F} + \mathbf{S})^{-1} \Delta\mathbf{u}} + \sigma_\epsilon^2 \delta(\Delta\mathbf{u}), \tag{14}$$

where C is a constant that is determined by σ_t, σ_n, σ_l, σ_s and σ_i as introduced in the previous sections, and σ_ϵ^2 denotes the noise variance.

For data interpolation, it is generally sufficient to use the normalized autocorrelation function $r(\Delta\mathbf{u}) = \rho(\Delta\mathbf{u})/\rho(\mathbf{0})$ which becomes

$$r(\Delta\mathbf{u}) = \underbrace{\frac{C}{C+\sigma_\epsilon^2}}_{\text{SNR}} \underbrace{e^{-\Delta\mathbf{u}^T(\mathbf{F}+\mathbf{S})^{-1}\Delta\mathbf{u}}}_{\text{Anisotropic decay}} \text{ for } \Delta\mathbf{u} \neq \mathbf{0}. \tag{15}$$

The autocorrelation function consists of two parts: a constant factor that is related to the Signal-To-Noise (SNR) level and a factor that describes how the correlation decays with spatial distance. The decay is determined by the metric tensor

$$\mathbf{M} = \mathbf{F} + \mathbf{S}\,, \tag{16}$$

i.e., a combination of the PFF and the ground surface structure. The metric describes an anisotropic correlation structure that depends on the local image edges and the current motion of the aircraft carrying the pushbroom sensor. This result is used in the next section to perform anisotropic interpolation of the pushbroom data.

4 Anisotropic scattered data interpolation

In section 2 it was established that pushbroom image rectification requires an interpolation from the irregular pushbroom samples onto a uniform grid. In section 3 it was furthermore established that the spatial dependence structure of pushbroom data is inherently anisotropic. In this section, different scattered data interpolation schemes known in the literature are adapted to take this local anisotropy into account. Five different methods are evaluated, one based on forward interpolation and four based on inverse interpolation. The pushbroom interpolation problem is illustrated in Fig. 5, where the value at the point $\mathbf{u}$ in the uniform output grid is predicted using the neighboring input samples $\mathbf{u}_1$, $\mathbf{u}_2$, ...

4.1 Anisotropic metric

To account for the fact that pushbroom data is differently correlated in different directions, we introduce the Mahalanobis distance metric

$$d_{\mathbf{M}}(\mathbf{u},\mathbf{v}) = \sqrt{(\mathbf{u}-\mathbf{v})^T\,\mathbf{M}^{-1}\,(\mathbf{u}-\mathbf{v})}, \tag{17}$$

where $\mathbf{u}$ and $\mathbf{v}$ are two arbitrary points in space and $\mathbf{M}$ is a metric tensor that defines a possibly anisotropic distance measure. The interpolation methods outlined below are valid for any choice of metric tensor, but for the pushbroom interpolation case we use the tensor defined in (16), which has its main axes oriented relative to a pushbroom line, i.e., along the normal and tangential directions $\hat{\mathbf{n}}_n$ and $\hat{\mathbf{n}}_t$. We introduce a shorthand notation for the distance from a pushbroom input sample point $\mathbf{u}_i$ to any arbitrary point $\mathbf{u}$

$$d_i(\mathbf{u}) = d_{\mathbf{M}_i}(\mathbf{u}_i,\mathbf{u})\,. \tag{18}$$

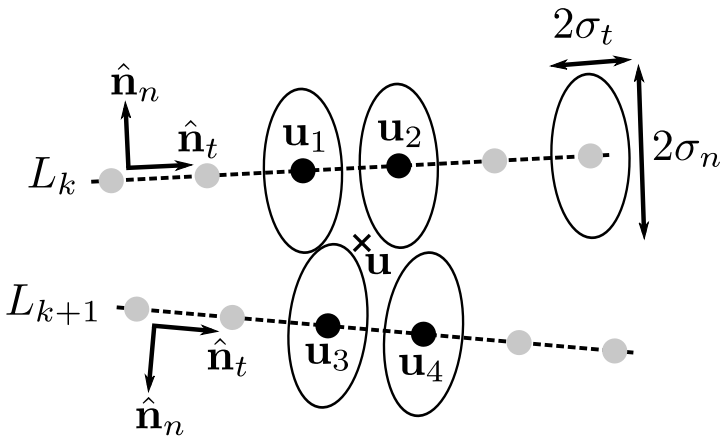


Figure 5: Samples on two consecutive pushbroom lines denoted by L_k and L_{k+1}. The goal is to predict the value at $\mathbf{u}$ given the measured values on the pushbroom lines $\mathbf{u}_1$, $\mathbf{u}_2$, ... The ellipses illustrate the anisotropic data dependence as a contour curve of a Gaussian function which is aligned with the corresponding pushbroom lines.

The orientation and degree of anisotropy defined by $\mathbf{M}_i$ vary between the pushbroom lines depending on the speed and trajectory of the carrier platform. Hence, the degree of anisotropy varies over the image. Therefore, if $\mathbf{M}_i$ and $\mathbf{M}_j$ are different, there is also an asymmetric distance relationship $d_i(\mathbf{u}_j) \neq d_j(\mathbf{u}_i)$.

4.2 Forward interpolation

In *forward interpolation* [8], also known as *Splatting* [26], each input pushbroom sample is spread (or splatted) out onto the neighborhood points in the uniform output grid. The contribution from each input sample point $z(\mathbf{u}_i)$, is accumulated iteratively in the output grid points

$$\mathbf{y}(\mathbf{u}) \leftarrow \mathbf{y}(\mathbf{u}) + \begin{bmatrix} w_i(\mathbf{u}) z(\mathbf{u}_i) \\ w_i(\mathbf{u}) \end{bmatrix}, \tag{19}$$

where $w_i(\mathbf{u})$ is a weight that depends on the distance between the irregular sample location $\mathbf{u}_i$ and the uniform grid location $\mathbf{u}$. Here we use the function

$$w_i(\mathbf{u}) = \begin{cases} e^{-d_i^2(\mathbf{u})} & \text{if } \mathbf{u} \in \mathcal{N}_K(\mathbf{u}_i) \\ 0 & \text{otherwise,} \end{cases} \tag{20}$$

which splats each input sample with a Gaussian anisotropic shape defined by the metric tensor $\mathbf{M}_i$, cf. (18). To limit the computational effort, each input sample is only splatted onto the output points in a square neighborhood $\mathcal{N}_K(\mathbf{u}_i)$ parameterized by the side K, i.e., the Gaussian in (20) is truncated. Here we set K so that at least 95% of the Gaussian is included within the splatting kernel:

$$e^{-\frac{K^2}{2\sigma_{max}^2}} < 0.05 \;\Rightarrow\; K > \sqrt{-2\sigma_{max}^2 \ln 0.05}\,, \tag{21}$$

where σ_{max} is the largest value of σ_n and σ_t. After the accumulation in (19), the interpolated value $\hat{z}(\mathbf{u})$ is found after normalization with the second element of $\mathbf{y}(\mathbf{u}) = [y_1(\mathbf{u}), y_2(\mathbf{u})]^T$:

$$\hat{z}(\mathbf{u}) = y_1(\mathbf{u})/y_2(\mathbf{u})\,. \tag{22}$$

That is, the predicted value $\hat{z}(\mathbf{u})$ is a weighted average of the contributions of the input samples.

The advantage of the forward interpolation is that it is computationally efficient. A main drawback is that holes with undefined values are created in the output image if the neighborhood $\mathcal{N}_K(\mathbf{u}_i)$ is too small.

4.3 Inverse interpolation

In an inverse interpolation scheme, an interpolated value is calculated as

$$\hat{z}(\mathbf{u}) = \sum_{\mathbf{u}_i \in \mathcal{N}(\mathbf{u})} w_i \, z\left(\mathbf{u}_i\right), \tag{23}$$

where w_i represent weights, $z\left(\mathbf{u}_i\right)$ is an input sample and $\mathcal{N}(\mathbf{u})$ denotes a neighborhood around $\mathbf{u}$. In principle, $\mathcal{N}(\mathbf{u})$ can encompass the entire space so that all input samples contribute to the interpolated value, but smaller localized neighborhoods are generally chosen. In this work, the nine or four closest input samples are typically used, see Fig. 5. The weights w_i are determined as a function of distance (17). Without loss of generality, let us assume that we found n input points in the neighborhood $\mathcal{N}(\mathbf{u})$ and denote them $\mathbf{u}_1, \ldots, \mathbf{u}_n$. Different methods have been suggested for choosing the weights $w_1, \ldots, w_n$ in (23). In this work, the *Nearest Neighbor*, *Inverse Distance Weighted*, *Natural Neighbors*, and *Kriging* interpolation schemes are extended to anisotropic interpolation and compared for the pushbroom image rectification.

The inverse interpolation scheme guarantees that each point in the output grid is assigned a predicted value. The main disadvantage of inverse interpolation schemes for irregular inputs is the computational complexity of determining the neighboring input points $\mathbf{u}_i \in \mathcal{N}(\mathbf{u})$. This section is concluded with a trick that exploits the semi-structured nature of pushbroom data to reduce the computational workload by many orders of magnitude. It can also be noted that inverse interpolation schemes are trivially parallelized as the points in the output grid are calculated independently of each other through (23).

4.3.1 Nearest Neighbor interpolation

In NN interpolation, only the closest input sample is considered,

$$\min_i d_i\left(\mathbf{u}\right), \tag{24}$$

for which the weight is set to 1 in (23). Note that the anisotropic distance from the input samples is used. Another characteristic of the NN interpolation is that the same neighbor will be the closest regardless of the size of the isotropic component of the metric tensor $\mathbf{M}$; only the anisotropic part of $\mathbf{M}$ can yield a different neighbor.

4.3.2 Inverse Distance Weighted interpolation

In the IDW interpolation, also known as Shepard's method [21], the weights in (23) are set to

$$w_i = \frac{\gamma}{d_i\left(\mathbf{u}\right)^2}, \tag{25}$$

where γ is a normalization factor chosen so that $\sum w_i = 1$. Just as for the NN interpolation, only changes to the anisotropic part of $\mathbf{M}$ give different interpolation results. Changes to the isotropic magnitude of $\mathbf{M}$ will not change the weights w_i.

4.3.3 Natural Neighbors interpolation

Natural neighbors [5, 1] (NAT) is a technique originally developed for solving partial differential equations on irregular grids [5]. In most inverse interpolation schemes, the weights in (23) are based on distances. NAT instead uses an area-based measure to compute the weights. The first step of the algorithm is to triangulate all input samples using the Delaunay algorithm, from which then the Voronoi tessellation is computed, as shown in Fig. 6 left. Next, the output interpolation point $\mathbf{u}$ is added to the set and

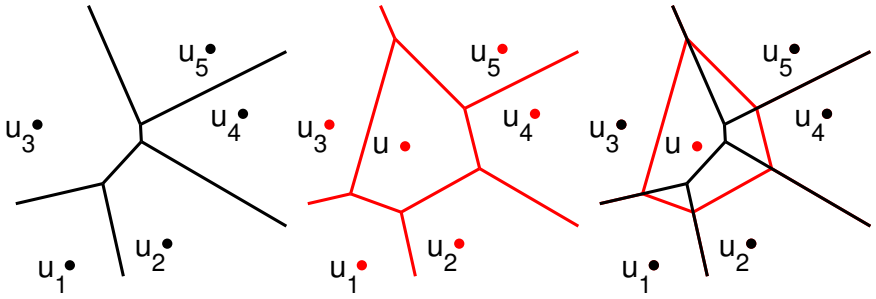


Figure 6: Illustration of natural neighbors interpolation. Left to right: Voronoi cells for $\{\mathbf{u}_i\}$, Voronoi cells for $\{\mathbf{u}_i\} \cup \mathbf{u}$, intersection of the two.

the Voronoi tessellation is updated, as shown in Fig. 6 middle. Finally, the intersection of the regions in the two tessellations is used to find areas

$$a_i = \texttt{area}(\texttt{region}(\mathbf{u}_i) \cap \texttt{region}(\mathbf{u}))\,, \tag{26}$$

one for each input data point, as shown in Fig. 6 right. Using these, the weights in (23) are computed as:

$$w_i = a_i / \left(\sum_{\mathbf{u}_k \in \mathcal{N}(\mathbf{u})} a_k \right) . \tag{27}$$

The use of an area-based weight computation in NAT was introduced to better handle highly irregular sampling densities [1]. For example, having many input samples on one side of the output sample will lead to a bias which area based approaches automatically compensates for. As NAT relies on an initial triangulation with a global metric,

there is no straightforward modification of the NAT method to make it take a local anisotropic metric into account.

4.3.4 Kriging interpolation

Kriging interpolation in general uses the covariance function $\rho(\mathbf{u}_i, \mathbf{u}_j)$ between sample locations to derive the optimal weights in (23) in a Best Linear Unbiased Estimator (BLUE) sense [11]. The covariance between two points typically decreases with the spatial distance so that the covariance also can be seen as a measure of distance between points. The so-called ordinary Kriging equation is used here for finding the interpolation weights:

$$\begin{bmatrix} w_1 \\ \vdots \\ w_n \\ -\mu \end{bmatrix} = \begin{bmatrix} \rho(\mathbf{u}_1, \mathbf{u}_1) & \cdots & \rho(\mathbf{u}_1, \mathbf{u}_n) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \rho(\mathbf{u}_n, \mathbf{u}_1) & \cdots & \rho(\mathbf{u}_n, \mathbf{u}_n) & 1 \\ 1 & \cdots & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} \rho(\mathbf{u}_1, \mathbf{u}) \\ \vdots \\ \rho(\mathbf{u}_n, \mathbf{u}) \\ 1 \end{bmatrix} \tag{28}$$

The covariances between the input points $\rho(\mathbf{u}_i, \mathbf{u}_j)$ capture the sampling density, and the covariances with the interpolation point $\rho(\mathbf{u}_i, \mathbf{u})$ can be interpreted as distances. The additional parameter μ is a nuisance parameter that is not used but which is required in (28) to ensure that $\sum w_i = 1$. A Gaussian covariance function that uses the anisotropic metric defined in (18) is used here as follows:

$$\rho(\mathbf{u}_i, \mathbf{u}_j) = e^{-d_i^2(\mathbf{u}_j)}. \tag{29}$$

4.4 Parameter tuning for pushbroom interpolation

To make a fair comparison of the different interpolation methods, the parameters of each method are tuned using a procedure described in this section. The tuned parameters include σ_i which controls the surface structure covariance, as well as σ_n and σ_t which control the pixel footprint size. These parameters are grouped into a parameter vector $\mathbf{p} = (\sigma_i, \sigma_n, \sigma_t)$. Optimal parameter values are determined using a cross-validation scheme: For a certain choice of $\mathbf{p}$, one actual sample $z(\mathbf{u}_i)$ is removed from the pushbroom data set. The predicted value $\hat{z}(\mathbf{u}_i)$ at this position is then interpolated using the remaining samples in the data set. By repeating this for many samples the interpolation accuracy can be measured. In this work, the relative error is used as accuracy measure

$$\varepsilon(\mathbf{p}) = \frac{1}{|\mathcal{E}|} \sum_{i \in \mathcal{E}} \frac{|\hat{z}(\mathbf{u}_i) - z(\mathbf{u}_i)|}{z(\mathbf{u}_i)}, \tag{30}$$

where $|\mathcal{E}|$ is the size of the evaluation set. Using this cross-validation error, one can search for the parameter vector $\mathbf{p}^*$ that gives the best interpolation accuracy. To find values close to the optimal ones, a brute-force grid search in reasonable intervals of each parameter is carried out. The final tuning is then made using a non-linear Nelder-Mead simplex optimization with the parameters found in the coarse search as starting point.

The parameters $\{\sigma_i,\ \sigma_n,\ \sigma_t\}$ do not have independent influences on the accuracy measure in (30). Specifically, the surface structure component $\mathbf{S}$ and the pixel footprint component $\mathbf{F}$ will both likely contain an isotropic part. The parameter optimization described above may therefore choose to put the isotropic part in $\mathbf{S}$ by adjusting the magnitude of σ_i, or in $\mathbf{F}$ by adjusting the magnitudes of σ_n and σ_t, i.e., the parameter set $\{\sigma_i, \sigma_n, \sigma_t\}$ results in almost the same structural covariance as $\left\{\sqrt{\sigma_i^2 + a}, \sqrt{\sigma_n^2 - a}, \sqrt{\sigma_t^2 - a}\right\}$, as long as $a < \sigma_n^2$ and $a < \sigma_t^2$. Note that the combined covariance $\mathbf{M} = \mathbf{F} + \mathbf{S}$ in (16), which ultimately is used for the interpolation, remains the same with either choice, so this becomes a problem of parameter interpretability rather than a problem of using the found parameters for the rectification. Nevertheless, one should use sample points at strong surface structures for the cross-validation in the optimization, as these carry more information about $\mathbf{F}$ and $\mathbf{S}$ than samples on isotropic surfaces. In practice, we do this by sorting the pixels in $\mathcal{E}$, according to their gradient strength, i.e., the trace of the structure tensor $\mathrm{tr}(\mathbf{T})$ computed according to (9). We then use only the 10% of pixels with the largest gradient values during parameter tuning. When reporting evaluation scores, however, the entire evaluation set $\mathcal{E}$ is used.

Finally, to compare results of isotropic and anisotropic interpolation, parameters for purely isotropic interpolation that do not account for surface structure of anisotropic footprints are also optimized. For this optimization, σ_n and σ_t are set to zero, and σ_i is used to define $\mathbf{S}$ according to (12).

5 Computational speedup for inverse interpolation of pushbroom data

In regular image resampling on uniform grids, the neighbors in $\mathcal{N}(\mathbf{u})$ are immediately given by the grid structure. When the input is irregularly sampled, a naive implementation must compute the distance from the interpolation point $\mathbf{u}$ to *all* input points $\mathbf{u}_i$ to determine the neighbors in $\mathcal{N}(\mathbf{u})$. If the number of input points is large this becomes computationally very expensive. Below, an approach is presented that exploits the semi-regular structure of pushbroom data to speedup the neighbor-finding procedure. The key idea is to approximate each line of pushbroom samples L_k in a parameterized standard line equation form

$$L_k : A_k x + B_k y + C_k = 0. \tag{31}$$

The parameters A_k, B_k and C_k can be found by a least-squares fit or simply by connecting the first and last points on the pushbroom line. Using this parameterization, we can efficiently

1. Identify the closest pushbroom line to $\mathbf{u}$.
2. Immediately predict the closest input samples on this line to $\mathbf{u}$ by assuming that the samples are equidistant on the line.

The shortest orthogonal distance $d(\mathbf{u}, L_k)$ from a point $\mathbf{u}$ to the line L_k is given by

$$d(\mathbf{u}, L_k) = \frac{|A_k u_x + B_k u_y + C_k|}{\sqrt{A_k^2 + B_k^2}}, \tag{32}$$

and the closest line(s) are thus easily identified. Next, the closest point on each parameterized line is also easily calculated. This closest point will in general not coincide with an input sample location, but as the input samples are almost evenly distributed on the pushbroom lines, the indexes of the input points that are closest to $\mathbf{u}$ can be predicted.

This trick makes it possible to quickly home in on a small set of candidate neighbor points. Thus, instead of calculating the distance to *all* input points to find the neighbors to $\mathbf{u}$, we need only calculate the distance to a handful of points. Specifically, if we have N_{lines} each consisting of $N_{samples}$ in a pushbroom data set, to interpolate one point, the straightforward naive implementation requires $\mathcal{O}(N_{lines}N_{samples})$ distance computations whereas the method above only requires $\mathcal{O}(N_{lines})$ distance computations. Since $N_{samples}$ typically is larger than 1000 samples, the method above increases the computational efficiency by three orders of magnitude.

It is stressed that once the candidate neighbor points have been found, the actual distances to these points are calculated for the interpolation using their georeferenced coordinates; the line parameterization is just used as an efficient way of identifying a small subset of neighbor candidate points. The parameters $\{A_k, B_k, C_k\}$ are calculated for all lines only once in a pre-computation step, and the computational effort for this is insignificant. It should be stressed that the above procedure works also when the samples on a pushbroom line deviate slightly from a straight line, e.g., due to lens distortion and small effects caused by the topography in the scene. It will also work if the pushbroom lines cross each other or if the ground prints of lines appear in reversed order relative to the flight direction due to strong motions caused by, e.g., banking or turbulence.

6 Data

Data used for evaluation in this work was acquired over the city of Oslo using the hyperspectral HySpex VNIR-1600 pushbroom sensor that is part of the FFI demonstrator system described in [22]. Each line in the image consists of 1600 pixels and 160 spectral bands are measured in the visual to near-infrared wavelengths. Each sensor element has an instantaneous field-of-view of approximately 0.18 mrad across- and 0.36 mrad along-track, yielding an average ground sample distance for this particular data set of about 35 cm. The data was georeferenced using auxiliary INS and DSM data [16]. Three subsets from the 8.6 km long swath were selected for the evaluations experiments, see Fig. 7. All three data sets have the same input size, i.e., 600 pushbroom lines times the 1600 samples of each line. The spatial resolution of the rectified output images $(\Delta x, \Delta y)$ was set to 30×30 cm^2 in all experiments. Table 1 summarizes the sizes of the input data and the uniform output grids for the three data sets. As the aircraft had different altitudes and velocities in different parts of the swath, the density of the irregular input grid is varying, and data set 3 is more sparse than the others.

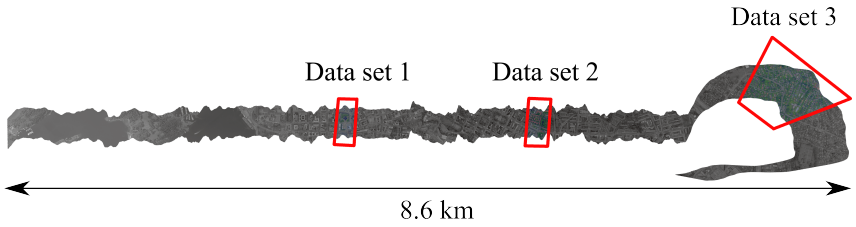


Figure 7: Three data sets for evaluation are taken from different parts of a long pushbroom swath over the city of Oslo. Each data set consists of the same number of pushbroom lines.

Table 1: Sizes of the input pushbroom data and the uniform output grid.

	Input size	Output size
Data set 1	600×1600	986×1338
Data set 2	600×1600	1034×1467
Data set 3	600×1600	3181×2098

7 Results

The interpolation methods described in Section 4 were implemented in C/C++, except for the NAT method for which the *TriScatteredInterp*-function in Matlab was used. Below, pushbroom image rectification using the different interpolation methods is compared in terms of accuracy, parameter stability and computational speed.

7.1 Parameter tuning

The parameters obtained with the cross-validation procedure outlined in Section 4.4 are presented in Table 2. The IDW method is not sensitive to the isotropic part of the metric tensor, as discussed in Section 4.3. For this reason, the optimization is not stable for this method and only σ_n was optimized, keeping σ_i and σ_t manually set to 1.0.

As discussed in section 4.4, there is an inherent ambiguity in the isotropic part of the covariance. In Table 2, we can see that this has caused most of the isotropy to end up in σ_i. While this has no negative effect on the resultant interpolation quality, it is still unfortunate, as the found parameters are more difficult to interpret. Still, the larger values of σ_i suggest that the anisotropic surface component is more important to model than the anisotropic pixel footprint function. However, σ_n and σ_t also get significant values, indicating that the model of an anisotropic footprint also contributes to the interpolation accuracy. In most cases we have $\sigma_n > \sigma_t$, which means that the pixel footprint is stretched in the movement direction of the aircraft. Dataset #3 is of a different nature than datasets #1 and #2, with the same number of samples covering a much larger area, see Fig. 7. This makes the optimizer prefer larger covariances for dataset #3, and as can be seen in Table 2 the larger covariance is produced by a large σ_i.

The cost function $\varepsilon(\mathbf{p})$ for the Splatting method is plotted in Fig. 8. To investigate

Method	Parameter	DS1	DS2	DS3	1% rise bracket
NN	σ_i	0.90	1.20	0.96	$[0.84, 0.98]$
	σ_t	0.65	0.00	0.35	$[0.50, 0.76]$
	σ_n	0.00	0.00	0.00	$[0.00, 0.20]$
IDW	σ_i	1.00*	1.00*	1.00*	-
	σ_t	1.00*	1.00*	1.00*	-
	σ_n	3.61	3.02	0.00	$[2.26, 5.09]$
Kriging	σ_i	0.65	0.72	1.88	$[0.48, 0.76]$
	σ_t	0.12	0.00	0.00	$[0.00, 0.36]$
	σ_n	0.46	0.30	0.00	$[0.30, 0.58]$
Splatting	σ_i	1.12	1.18	2.70	$[0.95, 1.27]$
	σ_t	0.00	0.00	0.00	$[0.00, 0.073]$
	σ_n	0.50	0.33	0.00	$[0.31, 0.65]$

Table 2: Optimized values for interpolation methods on the different datasets. 1% rise brackets are found on dataset 1. * means that the value was set manually.

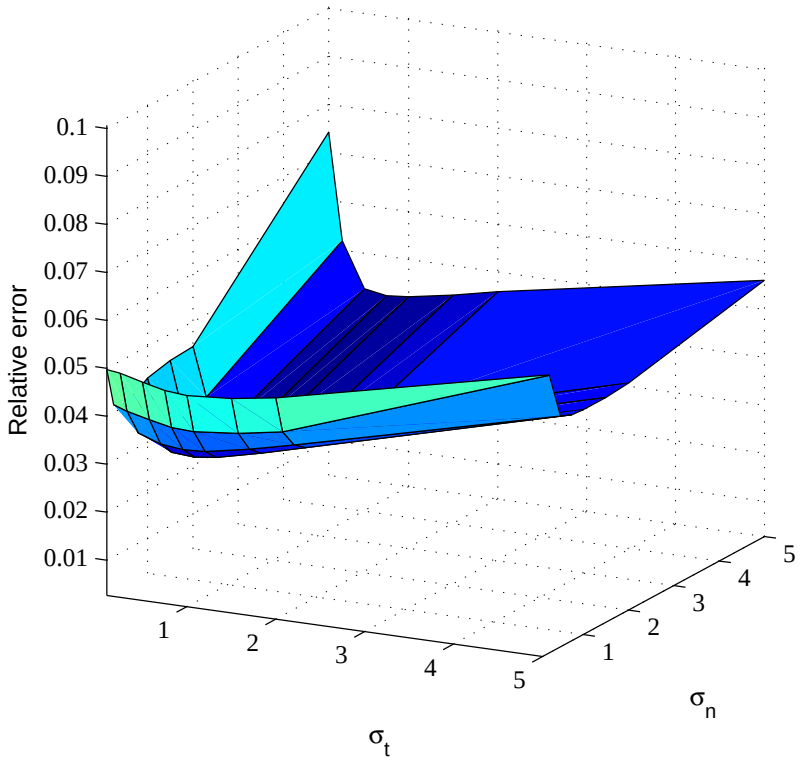


Figure 8: Example of a grid search. The relative interpolation error is plotted as a function of the σ_n and σ_t parameters (with $\sigma_i = 0$) in the Splatting method.

the stability of the parameters, we characterize the local shape of the cost function at the minimum $\varepsilon(\mathbf{p}^*)$ by checking how far we have to move in each direction for it to rise by 1%. These rise points can be found using the following approximative procedure: Very close to the optimum, the cost function (30) as a function of one of the parameters, may be approximated with a second order Taylor expansion. Thus, we first move a small delta h up and down from the optimum $\mathbf{p}^* = (\sigma_i^*, \sigma_t^*, \sigma_n^*)$, along each parameter dimension. We then fit quadratic polynomials to the function values and arguments along each axis. Using the fitted polynomials, we then find the points where $\varepsilon(\mathbf{p})$ has risen 1% above the minimum. These intervals are reported in the rightmost column in Table 2. For example, the parameters found on dataset #2 agree well with the ranges found on dataset #1. Thus, we conclude that the found parameters generalize well between these two sets.

7.2 Interpolation accuracy comparison

The average relative interpolation errors across all three datasets for each interpolation method with optimal parameters are plotted in Fig. 9. It is clearly seen that the NN method has inferior interpolation accuracy. The remaining methods all perform similarly, with only a small accuracy advantage of the Kriging method. It can also be seen that the anisotropic interpolation in all cases perform better than the isotropic interpolation. Fig. 10 shows the interpolation error per dataset. Again the NN method has the highest error while the other methods perform similarly. Data set 3 has somewhat higher interpolation errors due to the more irregular flight path and larger ground sampling distances, cf. Fig. 7.

In Fig. 11, three bands in the red, green and blue wavelengths have been rectified to produce RGB-images for a visual comparison between the methods. The NN method exhibits blocking artifacts that are typical for this method. For the remaining methods, a close examination reveals slight differences in the degree of smoothness, but overall they are similar. This is consistent with the interpolation accuracy results above. In some of the rectified images, the pushbroom line sampling pattern is visible, see for example the third and fourth rows from the top in Fig. 11. The line patterns are most prominent in the NN, Splatting and IDW methods, but for Kriging the effect is almost completely removed. To further visualize differences between the interpolation methods, all spectral bands in the hyperspectral datasets were rectified. In Fig. 12, the spectra in selected pixels in the evaluation set are plotted together with the actual measured spectra. Again the main difference is between the NN and the other methods.

7.3 Surface structure dependency

In order to further analyze the benefits of the anisotropic interpolation model, we have also separated the evaluation pixels in subsets with large and small amounts of surface structure respectively. All ground truth pixels are sorted according to their gradient strength as reflected in the trace of the structure tensor $\mathrm{tr}(\mathbf{T})$ computed according to (9), and the top 10% are denoted the ANISO subset. Similarly, the lowest 10% are denoted the ISO subset. Errors for these two subsets for dataset 2 are reported in Table 3 for the IDW and Splatting methods respectively. The largest difference between the

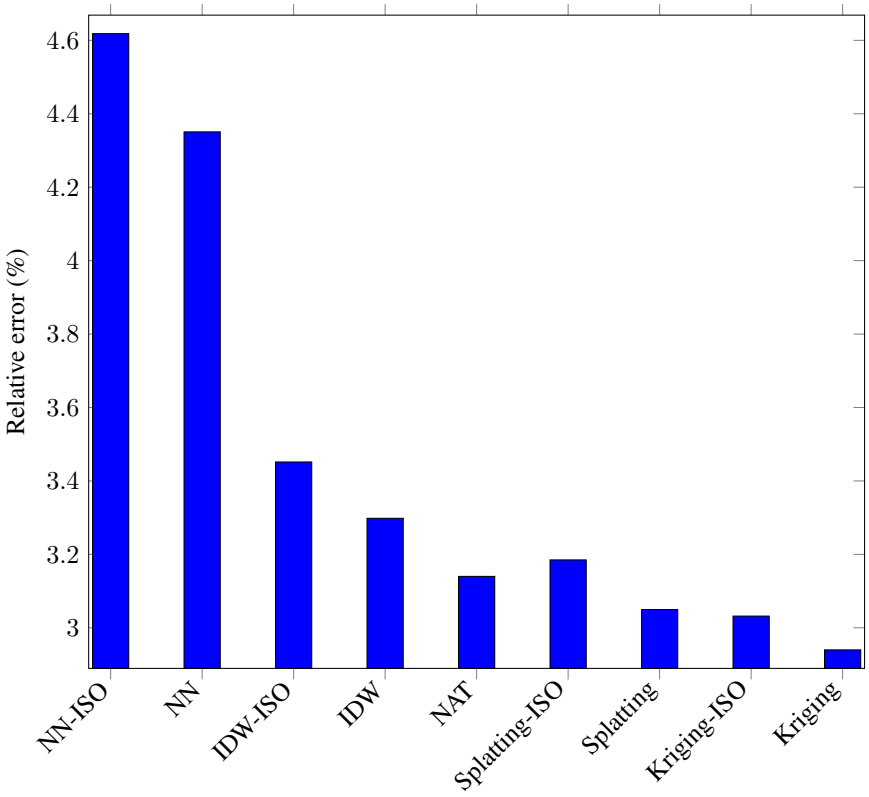


Figure 9: Accuracy comparison of the interpolation methods with tuned optimal parameters. The relative interpolation error has been averaged across all three datasets. The suffix ISO indicates isotropic interpolation and no suffix means anisotropic interpolation. The NN method has the lowest accuracy and the Kriging method the highest accuracy.

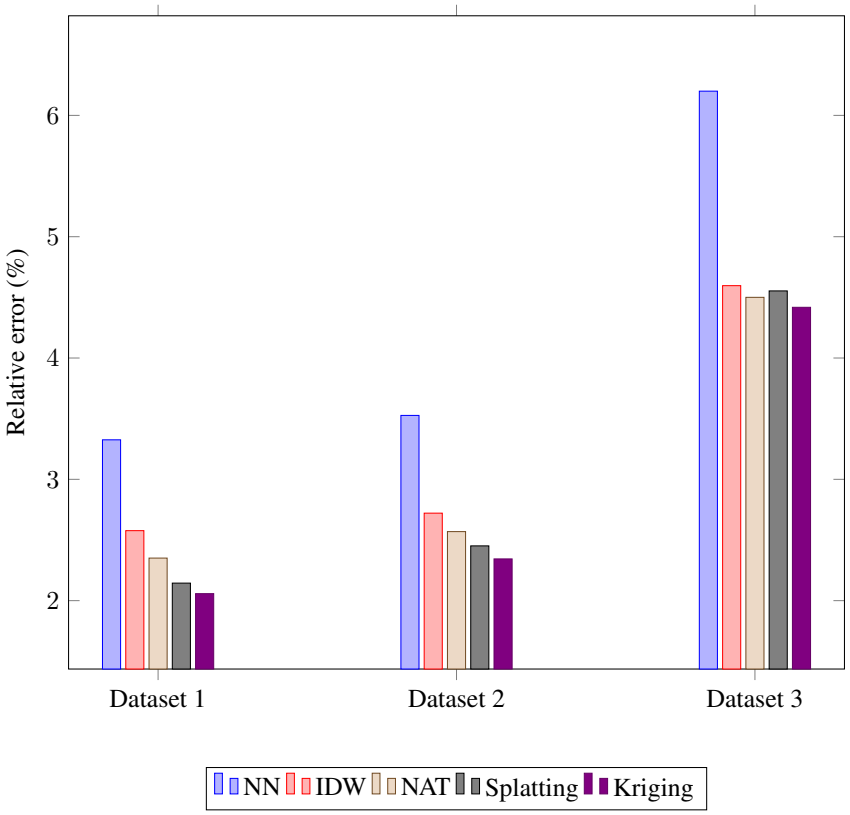
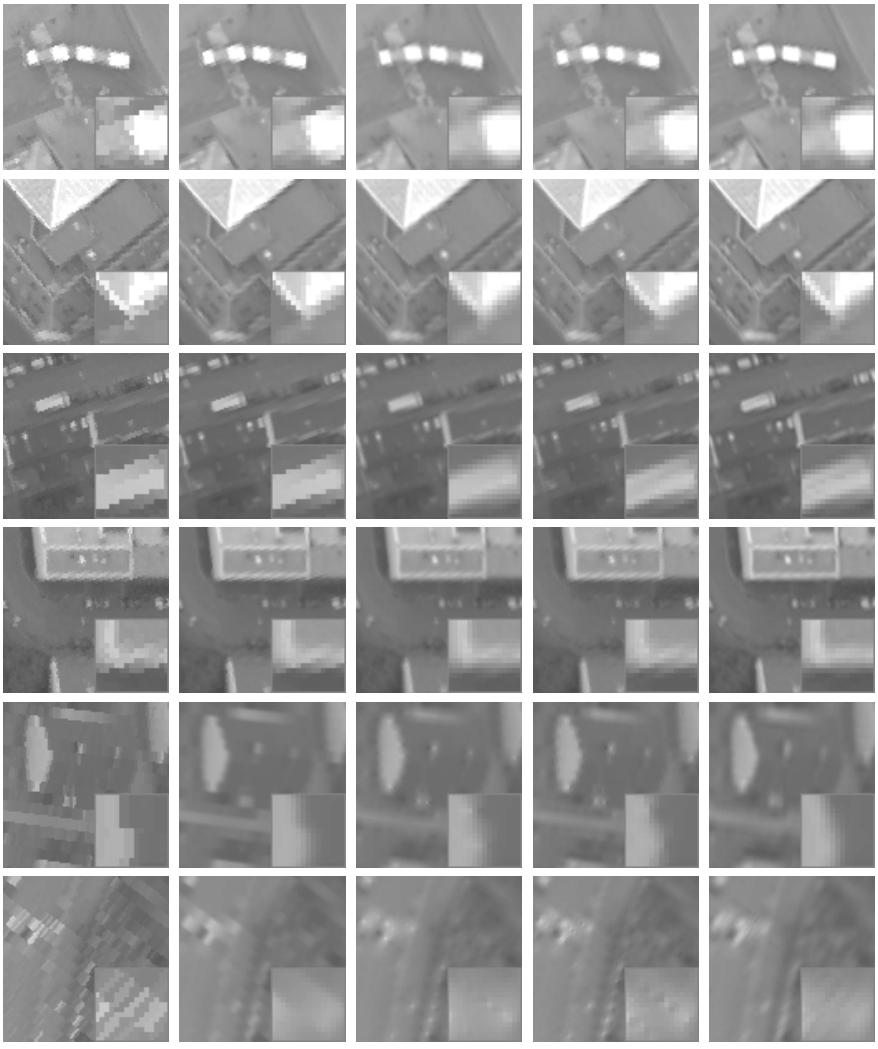


Figure 10: Accuracy results for each of the three datasets using the different tuned anisotropic interpolation methods.



(a) NN (b) Splatting (c) NAT (d) IDW (e) Kriging

Figure 11: Details of the reconstruction results. Top two rows are from dataset 1, middle two rows are from dataset 2 and bottom two rows are from dataset 3. The images are best viewed in the electronic version of this article.

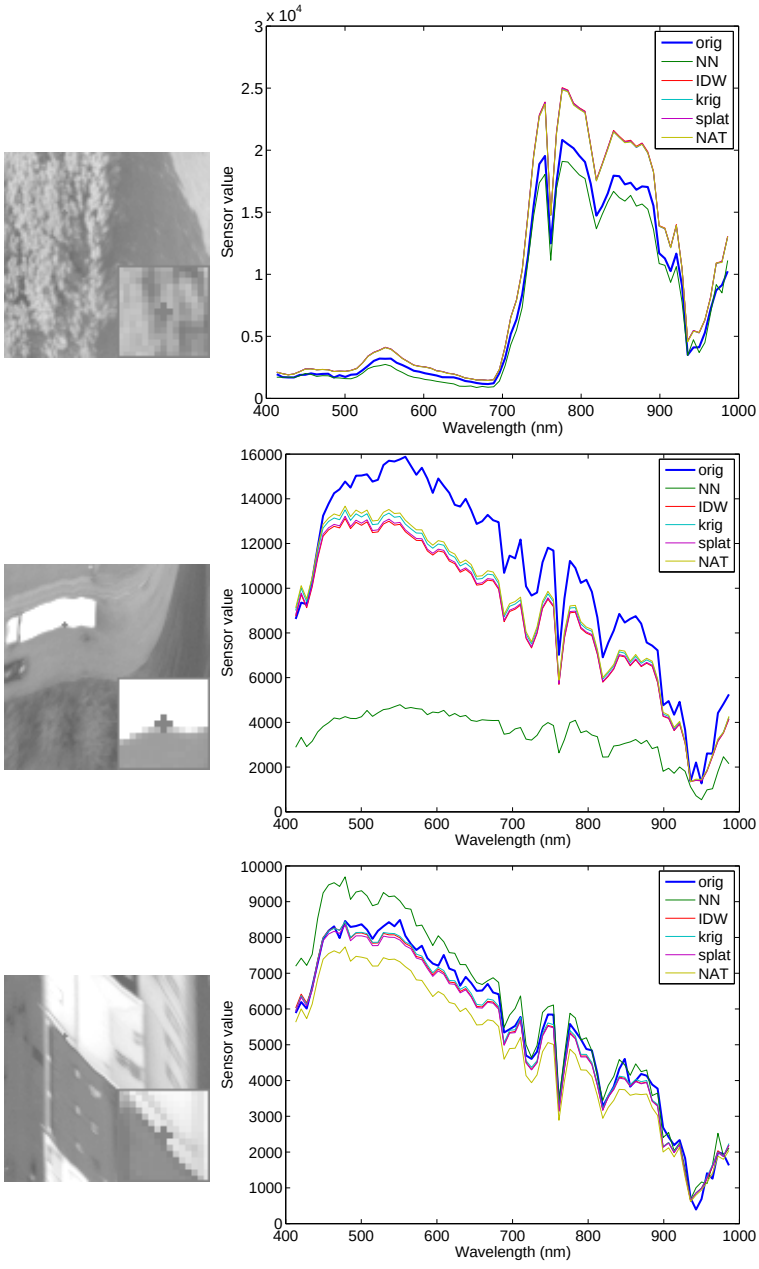


Figure 12: Interpolation for a few pixels in all spectral bands of the hyperspectral image. Left: the pixel used for interpolation. Right: Spectral plots for the different methods. "orig" is the actually measured spectrum and the other curves are different interpolation results.

IDW	ISO	ANISO
10% ISO	1.29%	1.28%
10% ANISO	7.13%	6.18%

Splatting	ISO	ANISO
10% ISO	1.29%	1.27%
10% ANISO	6.13%	5.11%

Table 3: Comparison of errors in regions with isotropic and anisotropic ground structure for the IDW and Splatting methods on dataset 2.

methods using anisotropic or isotropic surface structure model is for the ANISO subset. Similar results are obtained for the other interpolation methods.

7.4 Speed comparison

Depending on the application, computational speed may be an important factor. The interpolation methods have quite different computational complexities and we have compared the speed between the methods on the different datasets, see Table 4. The

Table 4: Method timings in seconds (serial execution)

	Dataset 1	Dataset 2	Dataset 3
Splatting	0.4	0.7	4.0
NN	5.4	6.3	26.3
IDW	5.5	6.5	27.5
NAT	21.4	25.8	141.4
Kriging	39.3	43.7	208.3

numbers in this table were obtained with serial implementations of the algorithms. The forward interpolation scheme represented by the Splatting method is about an order of magnitude faster than the other methods. Among the inverse interpolation schemes, NN and IDW are the fastest. The computational effort in these methods is mainly spent on finding the closest input samples for the interpolation according to the procedure outlined in Section 5. The Kriging interpolation requires additional computations, e.g., the matrix inverse in (28), and it is the slowest method. The NAT method requires a triangulation pre-processing step which is computationally demanding.

In the inverse interpolation schemes the points in the output grid are calculated independently of each other, cf. (23). The implementation of these schemes, in particular NN, IDW and Kriging, is for this reason trivial to parallelize. For example, on a CPU with 4 cores, a speedup of a factor 3.5 was obtained with a parallel implementation compared to the serial implementation. Thus, these methods benefit extensively from parallel processing architectures such as multicore CPUs or GPUs, and may outperform the Splatting method if such resources are available.

In summary, the Splatting method performs the interpolation with the lowest amount of computations, but with a proper implementation, all methods should be able to perform real-time, for example to rectify images on-line in a carrier aircraft.

8 Discussion

The image rectification problem addressed in this work is required for visualization of pushbroom data and for fusing results derived from such data with other imaging or geographic information. A key contribution is the modeling of the spatial dependence structure of pushbroom data in terms of the spatial covariance function. The covariance function involves two generally anisotropic and spatially non-stationary components: one that depends on the special properties of the pushbroom data acquisition and one that depends on the imaged surface structure itself. Based on this dependence model, five different interpolation methods for scattered spatial data are compared to interpolate the pushbroom samples at positions in a uniform grid.

In terms of interpolation results, the Nearest Neighbor method is inferior to the other interpolation methods, as can be expected due to its simplicity. The Kriging method consistently performs the best and has fewer visual striping artifacts, but the remaining methods are also viable from a practical view. Furthermore, the anisotropic interpolation schemes consistently yield better results than their isotropic counterparts. From a practical perspective, it is interesting to look at the computational performance of the different algorithms. In a straightforward implementation on a serial processing unit, the Splatting method has a clear advantage. With the trick to utilize the semi-structured sampling pattern of the pushbroom sensor, inverse interpolation schemes become feasible from a computational view, albeit still not as fast as the Splatting method. The inverse interpolation schemes have the attractive property of guaranteeing that there are no holes with undefined values in the rectified image. Moreover, the inverse interpolation schemes benefit trivially from parallel computational architectures, as each output pixel can be computed independently. With such resources available, inverse schemes could overtake the Splatting method.

A limitation of the current work, and a subject of future work is how to make the ground structure covariance estimation fully automatic. In this work we employ the approach suggested in [24], which uses neighborhoods of constant size, and thus implicitly assumes a constant sample density. In e.g. our dataset 3, where the samples are considerably more sparse, and irregular than in the other two, we had to adjust the neighborhood size manually, i.e. the σ parameter in (9).

A second limitation is that the footprint component of the covariance function is assumed to have constant magnitude (we only change its orientation), the reason being that more investigations are required of how to determine the parameters in the theoretical covariance model in (14) and to let it adapt from pixel to pixel. Through some approximations, e.g., a locally flat terrain around a pushbroom sample and a locally linear flight path, the pixel footprint function is modeled with an anisotropic Gaussian shape in this work. In recent work [17], the pixel footprint for each pushbroom sample was estimated using a Monte Carlo ray tracing method which takes the continuous flight path measured by the on-board INS system and a Digital Surface Model (DSM)

into account. Thousands of rays are sent to build up a non-parametric representation of the distribution of ground points that contribute to the pushbroom sample under consideration. Though computationally very demanding, the resulting distribution could be used for interpolation using a forward interpolation Splatting scheme.

Here we have only performed experiments with data acquired using a pushbroom sensor, but it may also be relevant for other types of sensors. Rolling shutter images acquired during motion need to be rectified and interpolated in order to be visualized correctly [20, 8] and any method used in this paper could be used for this, as long as there exists a mapping for all the pixels.

9 Conclusions

Different scattered data interpolation methods for the rectification of pushbroom images have been compared. A model of the pushbroom image acquisition process reveals an inherently anisotropic spatial data dependence structure that should be taken into account in the interpolation, in addition to an anisotropic surface model. This is supported by the experimental results, where consistent gains in accuracy were observed when adding anisotropic modeling, compared to the isotropic case. The anisotropic interpolation models presented here strikes a good balance between efficiency and accuracy, as it results in interpolation methods that can run at interactive speeds. Further gains in accuracy may be obtained with more sophisticated models, but at the expense of more demanding computations. To conclude with a recommendation of method choice, the Inverse Distance Weighted method strikes a good balance between accuracy and practical usability. The Kriging method is superior in terms of quality at the expense of a higher computational complexity, and the Splatting method could be the method of choice if computational resources are scarce.

Acknowledgement

This research has been funded by the Swedish Armed Forces and the CENIIT organization at the Linköping Institute of Technology.

References

[1] I. Amidror. Scattered data interpolation methods for electronic imaging systems: A survey. *J. Electron. Imaging*, 11(2), 2002.

[2] M. Andersson. *Controllable Multi-dimensional Filters and Models in Low-Level Computer Vision*. Linköping studies in science and technology. dissertations no. 282, Linköping University, 1992.

[3] Richard J. Aspinall, W. Andrew Marcus, and Joseph W. Boardman. Considerations in collecting, processing, and analysing high spatial resolution hyperspectral data for environmental investigations. *Journal of Geographical Systems*, 4(1):15–29, 2002.

[4] Fred L. Bookstein. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(6):567–585, 1989.

[5] Jean Braun and Malcolm Sambridge. A numerical method for solving partial differential equations on highly irregular evolving grids. *Nature*, 376(24):655–660, 1995.

[6] M. Demirhan, A. Ozpinar, and L. Ozdamar. Performance evaluation of spatial interpolation methods in the presence of noise. *Int. J. Remote Sens.*, 24(6):1237–1258, 2003.

[7] Ahmed A. Eldeiry and Luis A. Garcia. Using deterministic and geostatistical techniques to estimate soil salinity at the sub-basin scale and the field scale. In *Proceedings of the AGU Hydrology Days*, 2011.

[8] Per-Erik Forssén and Erik Ringaby. Rectifying rolling shutter video from hand-held devices. In *IEEE CVPR*, 2010.

[9] R. Franke. Scattered data interpolation: Tests of some methods. *Math. Comput.*, 38(157):181–200, 1982.

[10] R. Gupta and R.I. Hartley. Linear pushbroom cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):963–975, 1997.

[11] Edward H. Isaaks and R. Mohan Srivastava. *An Introduction to Applied Geostatistics*. Oxford University Press, 1989.

[12] Bernd Jähne and Horst Haussecker. *Computer Vision and Applications*. Academic Press, 2000.

[13] D. G. Krige. A statistical approach to some mine valuations and allied problems at Witwatersrand. Master's thesis, University of Witwatersrand, South Africa, 1951.

[14] S.-N. Lam. Spatial interpolation methods: A review. *Amer. Cartogr.*, 10(2):129–149, 1983.

[15] G. Matheron. The theory of regionalized variables and its applications, chapter 3. *Les Cahiers du Centre de Morphologie Mathématique de Fontainebleu*, 5, 1971.

[16] T. Opsahl, T. Haavardsholm, and I. Winjum. Real-time georeferencing for an airborne hyperspectral imaging system. In *Proceedings of the SPIE Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVII*, 2011.

[17] T. Opsahl and T. V. Haavardsholm. Estimating the pixel footprint distribution for image fusion by ray tracing lines of sight in a Monte Carlo scheme. In *SPIE 8743, Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XIX*, 2013.

[18] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer Verlag, 1997.

[19] J. Reguera-Salgado, M. Calvino-Cancela, and J. Martin-Herrero. GPU geocorrection for airborne pushbroom imagers. *IEEE Transactions on Geoscience and Remote Sensing*, 50(11):4409–4419, 2012.

[20] Erik Ringaby and Per-Erik Forssén. Efficient video rectification and stabilisation for cell-phones. *IJCV*, 96(3):335–352, 2012.

[21] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the ACM National Conference*, 1968.

[22] T. Skauli, T.V. Haavardsholm, I. Kåsen, G. Arisholm, A. Kavara, T. Olsvik-Opsahl, and A. Skaugen. An airborne real-time hyperspectral target detection system. In *SPIE 7695, Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVI*, 2010.

[23] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer Verlag, London, 2011.

[24] Hiroyuik Takeda, Sina Farsiu, and Peyman Milanfar. Kernel regression for image processing and reconstruction. *IEEE Transactions on Image Processing*, 16(2):349–366, February 2007.

[25] Joachim Weickert. *Anisotropic Diffusion in Image Processing*. ECMI Series, Teubner-Verlag, 1998.

[26] Lee Westover. Interactive volume rendering. In *CH Volume Visualization Workshop*, pages 9–16, 1989.

Chapter F

A Virtual Tripod for Hand-held Video Stacking on Smartphones

This is an edited version of the paper:

Erik Ringaby and Per-Erik Forssén. A virtual tripod for hand-held video stacking on smartphones. In *IEEE International Conference on Computational Photography*, Santa Clara, USA, May 2014. IEEE Computer Society.

A Virtual Tripod for Hand-held Video Stacking on Smartphones

Erik Ringaby and Per-Erik Forssén
Department of Electrical Engineering, Computer Vision Laboratory
Linköping University, Sweden

Abstract

We propose an algorithm that can capture sharp, low-noise images in low-light conditions on a hand-held smartphone. We make use of the recent ability to acquire bursts of high resolution images on high-end models such as the iPhone5s. Frames are aligned, or *stacked*, using rolling shutter correction, based on motion estimated from the built-in gyro sensors and image feature tracking. After stacking, the images may be combined, using e.g. averaging to produce a sharp, low-noise photo. We have tested the algorithm on a variety of different scenes, using several different smartphones. We compare our method to denoising, direct stacking, as well as a global-shutter based stacking, with favourable results.

1 Introduction

In this paper we propose an algorithm that can capture sharp, low-noise images in low-light conditions on a hand-held smartphone. Recent smartphone models such as the Apple iPhone5s, Acer Liquid S2 and Samsung Galaxy Note 3 have the ability to acquire bursts of high resolution images at a high rate. For smartphones equipped with a gyroscope sensor, such image bursts may be aligned or *stacked* at low cost, using sensor data. The stacked frames can then be fused into a single (less noisy) frame using e.g. averaging, median or bilateral filtering. This enables hand-held acquisition of sharp, low-noise images with long exposure times, without the requirement of additional mechanical image stabilisation hardware.

Video stacking on a smartphone currently requires a tripod, due to the *rolling shutter* (RS) distortions that appear during hand-held capture. In this paper we introduce an RS based correction, that allows also images from hand-held video to be stacked.

An illustration of the proposed approach is given in Figure 1. Instead of using one long exposure, with resultant blurring, many short exposures are used in sequence. When the photographer has a *static aim* (i.e. tries to aim at a fixed point in space), these individual exposures tend to have blur smears in a random distribution of directions. This means that when the frames are aligned (using gyroscope readings and image feature tracking), we obtain an effective *point spread function* (PSF) that is much more compact than one from a single long exposure.

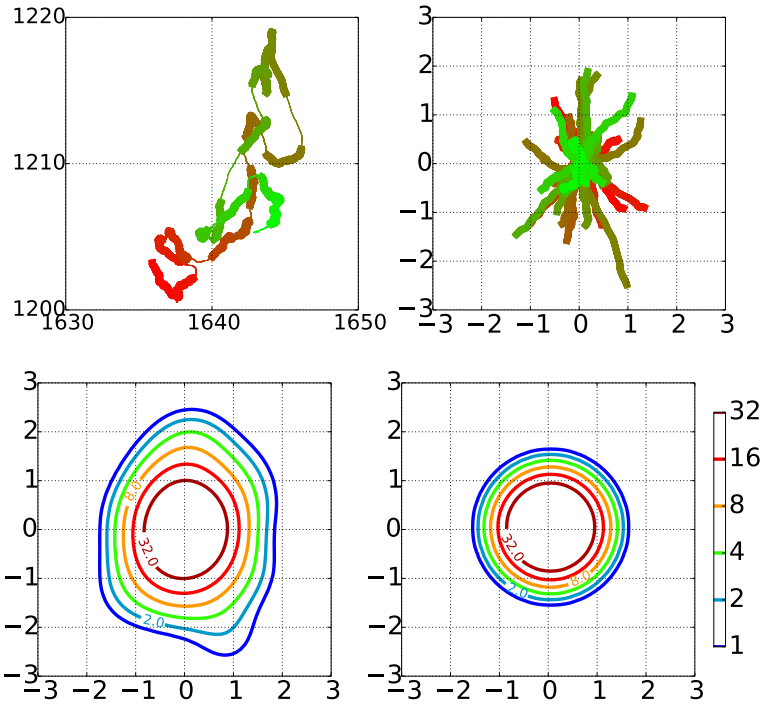


Figure 1: Illustration of video stacking idea. Top left: Trace of central pixel of an iPhone5 during hand-held capture of 20 frames at 14 fps, with 40 msec exposure time. The motion has been recorded with an L3G4200D gyroscope at 800Hz, and colours indicate time, ranging from red to green. Thick segments indicate individual exposures. Top right: Alignment of the exposure segments. Bottom left: Iso contours of the effective PSF (scaled to sum to 255), obtained by convolving the aligned exposure segments with the stationary PSF (here assumed to be a Gaussian with $\sigma = 0.5$). Bottom right: corresponding iso contours for a Gaussian with $\sigma = 0.5$.

1.1 Related work

Besides the stacking approach used in this paper, there are several other approaches to low-light image capture. One is to use pairs of flash and no-flash images, see e.g. [14] and [3], and another is to use a pair with one blurry and one noisy image, e.g. [22]. As these approaches rely on accurate alignment of frames, they could also benefit from the rolling shutter aware alignment procedure proposed here, if they were to be used on mobile devices.

Another related approach is video denoising. Here the exposure time is set low enough to obtain sharp, but noisy images, and then spatiotemporal denoising [11, 15] is performed. These methods find correspondences across several frames, using e.g. dense optical flow, and approximate KNN search, and this makes them infeasible to implement on a smartphone.

A much faster, but less accurate approach than video denoising is to denoise a single frame captured with a short exposure, using e.g. anisotropic diffusion [20]. We will compare to single frame denoising in the experiment section.

Single frame deblurring using inertial sensors (INS) has been investigated in [8]. Recently this has also been extended to use a rolling shutter camera model [19]. Such algorithms are iterative in their nature, as they obtain the final sharp images using e.g. Richardson-Lucy style deconvolution [16, 13]. If the sensor biases also need to be found, this requires a second optimisation loop outside the first one [8]. In general, these algorithms are either relatively efficient, but prone to ringing artifacts, or very expensive if complex priors are employed.

Single frame deblurring without INS is also an option, see e.g. [21], and the recent approach in [23]. In addition to performing deconvolution, these approaches also need to find the point-spread functions in each image location, and as this is typically done using a second optimisation loop outside the first one, these methods are an order of magnitude more expensive than methods that use INS data.

As nearly all digital video recording devices use rolling shutters, the video stacking is related to video stabilisation under rolling-shutter [17, 5]. Stabilisation of RS video has also been done using inertial sensors [6, 9]. Basically, the problem in video stacking is a video stabilisation, where the desired camera trajectory is a single point in space with static aim. In order not to introduce blurring however, stacking has a much higher requirement on stabilisation accuracy than video stabilisation. For high accuracy we use the cumulative quaternion B-splines introduced by Kim et. al [10], to interpolate the gyro samples. These splines were also recently used in an optimisation framework for SLAM in [12], by minimising the reprojection errors on tracked features and sensor data. Compared to [12] our algorithm is many orders of magnitude faster, as our optimisation only needs to solve for 4 unknowns, instead of several thousands.

Stacking has been popular for quite some time in astrophotography, and the idea actually originates here [1]. Here the camera is typically mounted on a telescope that tracks slow motion (e.g. of the moon, or the celestial sphere), and long exposure times are used. The motion to be compensated for is thus the residual of the tracking and the actual motion, and not the complex atmospherical aberrations observed at short time-scales.

Recently stacking using inertial sensors has been introduced by compact camera manufacturers, in e.g. Sony Cyber-Shot DSC RX100 [2]. These devices use the motion sensors to select a few frames (up to 6 for Sony) with low amounts of motion. These are then stacked with global frame alignment (as the camera appears to use the mechanical shutter this is justified), using inertial sensors. In contrast, the method in this paper uses a rolling shutter distortion model, and is able to make use of *all* frames in the acquisition interval. It thus has a much better light collection efficiency, and better noise suppression.

1.2 Video stacking

The method proposed in this paper makes use of the built-in gyro sensor on a smartphone, and a sparse set of tracked image features to stack frames acquired using rolling

shutter style exposure. The stacked frames can then be fused into a single (less noisy) frame using e.g. averaging, median or bilateral filtering.

For low-noise low-light photography, we should collect as much light as possible during the time the shutter is open. This means that for stacking, the *light collection efficiency*, e, is an important performance metric. This measure is approximately equal to the product of the shutter speed s in seconds, and the frame rate r in Hertz. For an N frame stack it is defined as the effective exposure time $s_e = sN$ divided by the time required to acquire the stack $T = (N-1)/r + s \approx N/r$.

$$e = s_e/T \approx sr. \tag{1}$$

For a single frame we always get $e = 1$, but e.g. a shutter speed of $s = 1/30$ sec and $r = 20$ fps, gives us $e \approx 0.67$.

Just like in classical photography, we have a built-in trade-off here: in order to eliminate motion blur, the shutter speed s should be short, but in order to collect more light it should be as long as the frame rate permits.

In order to improve the light collection efficiency (1) we will in general allow some motion blur. The rationale for this is illustrated in Figure 1. When the photographer has a static aim, the blur directions in consecutive images will be randomly distributed (see Figure 1 and 4), and this means that the final effective blur kernel will be much smaller than the smear in individual frames. The example in Figure 1 shows the effective PSF for stacking of 20 frames with 40 msec exposure time. Even though the smear length is about 3 pixels in each frame, the effective PSF is similar to a Gaussian of $\sigma = 0.5$.

2 Motion Model

We make use of a motion model that consists of a time continuous 3D rotation, and a frame global 3D translation. These models are estimated and applied in corrective fashion, one after the other. Such an approach normally requires *alternating optimisation*. The reason this works in one shot here is that the rotation model makes use of gyro sensors to estimate the rotation, and visual tracking to estimate the gyro bias and the time delay between gyro and camera. As the gyro sensors only sense rotation and not translation, the translation will not interfere with the rotation compensation, and the two models need only to be estimated once.

2.1 Rotation model

We use the 3D rotation model introduced in [4]. In this model, a point in the first frame, expressed in homogeneous coordinates as $\mathbf{x} = (x_1\ x_2\ 1)^T$, is related to its position in a subsequent frame $\mathbf{y} = (y_1\ y_2\ 1)^T$ according to:

$$\mathbf{x} \sim \mathbf{K}\mathbf{R}(t_\mathbf{x})\mathbf{R}^T(t_\mathbf{y})\mathbf{K}^{-1}\mathbf{y}\,. \tag{2}$$

Here $\mathbf{K}$ is the intrinsic camera matrix, $\mathbf{R}(t)$ is the time-continuous camera rotation to be estimated, and $\sim$ denotes equality up to scale. The times $t_\mathbf{x}$ and $t_\mathbf{y}$ correspond to when the image points $\mathbf{x}$ and $\mathbf{y}$ were observed.

2.2 Translation model

In [17] the authors found that the rotation model was good for rolling-shutter rectification since rotation is the dominant cause for the distortions. This model is used pairwise on neighbouring frames, but for a global alignment between all the frames, we also take translations into account. For the translations, we approximate the scene with a fronto-parallel plane. A point $\mathbf{y}$ in one of the frames may be re-projected onto this scene plane as $\mathbf{u}$ using:

$$\mathbf{u} = \lambda \mathbf{K}^{-1}\mathbf{y} = \lambda \begin{pmatrix} u_1 & u_2 & 1 \end{pmatrix}^T . \tag{3}$$

Now we may add a 3D displacement $\mathbf{d} = \begin{pmatrix} \Delta X & \Delta Y & \Delta Z \end{pmatrix}^T$, and re-project the result into the first image:

$$\mathbf{x} = \mathbf{K}(\lambda \mathbf{K}^{-1}\mathbf{y} + \mathbf{d}) = \begin{pmatrix} y_1 s + a \\ y_2 s + b \\ 1 \end{pmatrix} , \tag{4}$$

where $\{s, a, b\}$ are functions of the elements of $\mathbf{K}$ and $\mathbf{d}$. We may thus estimate $\{s, a, b\}$ instead of $\mathbf{d}$. Estimation of $\{s, a, b\}$ can be done efficiently from a set of corresponding points using least squares. This is the 2D equivalent of Horn's rigid motion estimation method [7].

3 Interpolation of Rotations

In order to obtain a smooth representation of the continuous rotation $\mathbf{R}(t)$, we use the cumulative quaternion splines proposed in [10]. A B-spline curve defined by knots $\mathbf{p}_k \in \mathbb{R}^n$ is evaluated as:

$$\mathbf{p}(t) = \sum_{k=0}^{K} \mathbf{p}_k B_k(t) . \tag{5}$$

The cumulative form of (5) is:

$$\mathbf{p}(t) = \mathbf{p}_0 \tilde{B}_0(t) + \sum_{k=1}^{K} \Delta\mathbf{p}_k \tilde{B}_k(t) \text{ , where} \tag{6}$$

$$\Delta\mathbf{p}_k = \mathbf{p}_k - \mathbf{p}_{k-1} \quad \text{and} \quad \tilde{B}_k = \sum_{l=k}^{K} B_k . \tag{7}$$

In analogy with this, cumulative splines on the rotation manifold may be defined, using unit quaternion knots, $\mathbf{q}_k = (\cos\alpha_k, \hat{\mathbf{n}}_k \sin\alpha_k) \in \text{Spin}(3)$, and quaternion operations as [10]:

$$\mathbf{q}(t) = \mathbf{q}_0^{\tilde{B}_0(t)} \prod_{k=1}^{K} \exp(\boldsymbol{\omega}_k \tilde{B}_k(t)) \text{ where} \tag{8}$$

$$\boldsymbol{\omega}_k = \log(\mathbf{q}_{k-1}^{*}\mathbf{q}_k) . \tag{9}$$

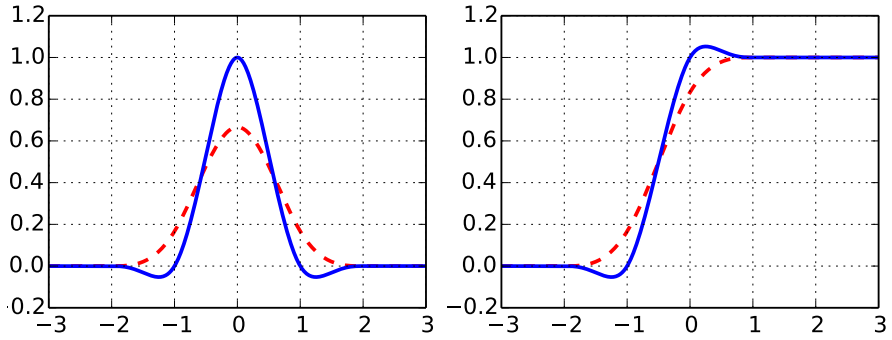


Figure 2: Kernels. Left: Interpolating quartic spline (solid blue) and B-spline (dashed red). Right: The corresponding cumulative kernels.

Here $*$ denotes the quaternion conjugate, and for a unit quaternion $\mathbf{q}$, the logarithm is defined as:

$$\log \mathbf{q} = \log\left(\cos\alpha, \hat{\mathbf{n}}\sin\alpha\right) = (0, \alpha\hat{\mathbf{n}})\,, \tag{10}$$

and $\exp()$ is the corresponding inverse operation. In [10] B-spline kernels were used, and these define a curve by approximation. As we are interested in interpolation, we will also try replacing the B-spline kernels with the classical interpolating cubic spline (with the common choice of $\partial B/\partial t(1) = -0.5$, see e.g. [18]), as well as the following *quartic* spline:

$$B_{\text{int}}(t) = \begin{cases} -\frac{1}{2}t^4 + \frac{5}{2}|t|^3 - 3t^2 + 1 & \text{if } |t| < 1 \\ \frac{1}{2}t^4 - \frac{7}{2}|t|^3 + 9t^2 - 10|t| + 4 & \text{if } |t| \in [1,2] \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

This spline is the unique piecewise quartic that satisfies the following 10 constraints: constant sum (1dof), $B : [-2,-1,0,1,2] \rightarrow [0,0,1,0,0]$ (5dof), $\partial B/\partial t$ and $\partial^2 B/\partial t^2$ continuous at $t = 1$ (2dof). $\partial B/\partial t : [0,2] \rightarrow [0,0]$ (2dof).

Figure 2 shows a B-spline kernel and the kernel defined in (11), and their corresponding cumulative kernels.

Note also that it is possible to move smoothly between interpolation and approximation by blending the interpolating and the approximating kernels:

$$\tilde{B}(t,\gamma) = \gamma\tilde{B}_{\text{int}}(t) + (1-\gamma)\tilde{B}_{\text{approx}}(t)\,. \tag{12}$$

Here $\tilde{B}_{\text{approx}}(t)$ is the B-spline kernel, and γ is a blending parameter.

3.1 Integration

Interestingly, the tangent vectors $\boldsymbol{\omega}_k$ in (9) are closely related to angular velocities. This allows us to compute them from the gyro data $\{\mathbf{g}_k\}_0^K$, using the expression:

$$\boldsymbol{\omega}_k = (0, \Delta t(\mathbf{g}_k + \mathbf{g}_{k-1} - 2\mathbf{b})/2)\,, \tag{13}$$

where Δt is the gyro sample time, and $\mathbf{b}$ is the gyro bias vector to be estimated. The rationale for (13) is that for small angles (i.e. small Δt) it is a good approximation of trapezoidal integration on the manifold of rotations.

4 Parameter Optimisation

The use of gyro data together with a camera requires estimation of the time delay t_d between gyro samples and camera frame timestamps, as well as the three element gyro bias vector $\mathbf{b}$, see (13).

In [9] a calibration procedure that finds t_d and $\mathbf{b}$ is proposed, our approach is quite similar, but we have replaced an initial global-shutter geometric constraint with a geometry free rejection, and added a rolling-shutter geometry based rejection later on.

First the reprojection error of (2) is used to define residuals for correspondences between neighbouring frames. As we want to avoid imposing geometric constraints on the correspondences, we use cross-checking rejection on KLT-features, as suggested in [4]. We start with setting the gyro bias to the sample mean and estimate the time delay t_d using point correspondences from a few frames in the beginning of the sequence. The parameter that minimises the squared sum of the residuals is found using non-linear optimisation. After convergence of this optimisation, we obtain residuals that approximately follow a Gaussian distribution. We have found that better accuracy of the sought parameters can be obtained by removing correspondences with residuals beyond the 3σ limit at this stage. After removal of these correspondences, we optimise for both t_d and $\mathbf{b}$ using correspondences from the whole sequence until convergence.

4.1 Robust Estimation

After t_d and $\mathbf{b}$ have been found, we can apply the rotation model (2) to all points. We do this and resample the images using forward interpolation as suggested in [4].

After image resampling, we have a fairly well aligned stack, but if the imaged scene is close to the camera we still need to apply the 3D translation model from section 2.2. In order to do this, we again run a KLT-tracker between the first frame, and each successive frame, and remove outliers using cross-checking. We then use the found correspondences to estimate the translation model (4) within a RANSAC [18] loop. For the model (4) the minimal number of sample correspondences is 2. This, and the low number of remaining outliers together mean that RANSAC usually finds the correct model after just a few trials. Once a model with a large ratio of inliers has been found, we re-estimate the model using all inliers.

In the final result, we want to avoid any unnecessary blurring caused by resampling the images twice. The final correction is thus obtained by applying both the rotation model and the translation model to the original image coordinates, and then resampling the original frames using forward interpolation.

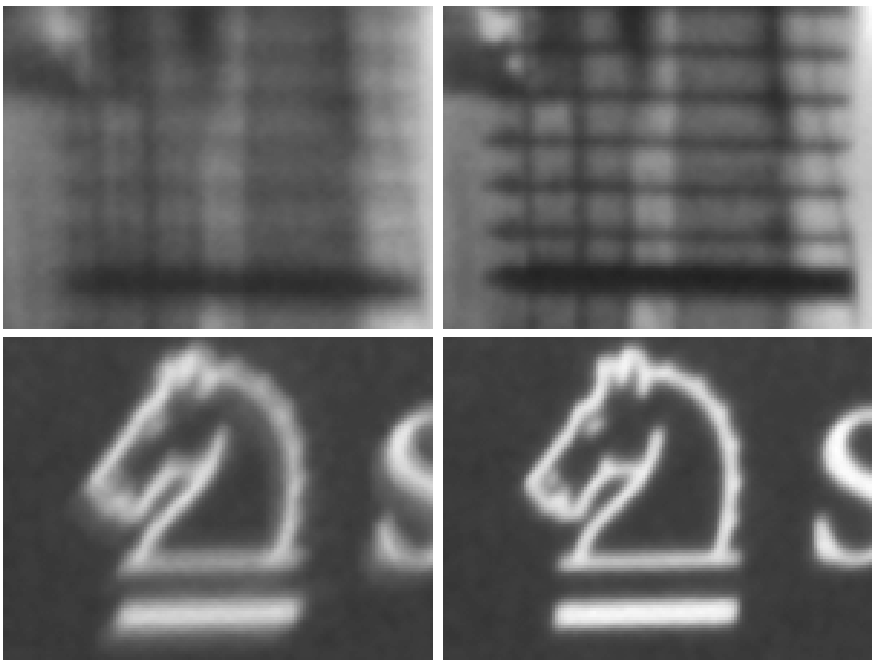


Figure 3: Zoomed in examples between global frame alignment (left) and our rolling shutter aware method (right). For full frames, see Figure 7, top left, and Figure 6, bottom left.

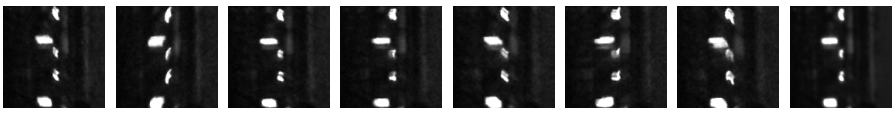


Figure 4: Columns 1-7: Example frames from a hand-held sequence showing different blurs. Right: Our result. (Best viewed electronically)

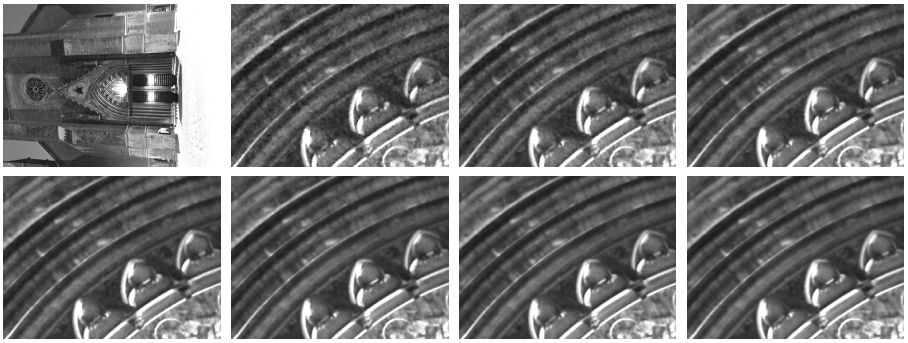


Figure 5: Scene captured using a physical tripod. Left to right, top to bottom: denoised image using 64 frames, zoomed in detail using 1, 2, 4, 8, 16, 32 and 64 frames respectively. (Best viewed electronically)

4.2 Algorithm Bottlenecks

The proposed algorithm is quite efficient. The current implementation is in Python, and runs on a PC, for ease of analysis. Currently, the most expensive part of the algorithm is image resampling and saving to disk, followed by optimisation, and feature tracking. Most of the time is currently spent on saving to disk and image re-sampling, but if resampling was to be done on the smartphone GPU, the cost of these steps would be negligible. Thus, an efficient implementation on a smartphone should be straight forward.

5 Experiments

We have tested our method on many real-life sequences and compare our results with the following methods: (1) the single frame denoising implemented in Photoshop CS 5.1[1], (2) direct averaging of the unaligned frames in the stack, (3) global frame alignment (i.e. without rolling shutter correction). In the experiments, our method uses 32 frames, and B-spline kernels, unless stated otherwise.

When capturing images in low light we usually get both motion blur and image noise. Since most of the motion blur is from rotation, the blur kernel is non-uniform, both across the frame [21] and temporally. In Figure 4 we give an example of how different it may look like across a sequence of frames from a hand-held sequence.

If we assume a rigid scene we could use as many frames as we want to obtain the final result, but since this is not always the case we have to trade the capture duration and the output noise level. In Figure 5 the noise level for stacks with increasing number of frames is shown. In this particular example, improvements beyond 16 frames are difficult to see.

5.1 Data collection

We have implemented an app for iOS that logs time-stamped full resolution frames, as well as gyro sensor data at 100 Hz. The obtained frame-rate is a function of the bus-speed, the chosen video-quality, and the computational power of the device. We have configured the app to record using the JPEG encoder, with quality set to 85%. This results in a recording speed of about 9 Hz on iPhone 4s, 14 Hz on iPhone 5, and 30 Hz on iPhone 5s. For the same amount of denoising, the 5s thus has a shorter stacking time, due to its superior light collection efficiency, see (1).

The five stacks used in this experiment have been made available in a public dataset[2]. This includes full resolution input images, frame timestamps, and logs from the builtin gyroscope.

[1] Note that Photoshop CC also has a Shake Reduction feature, which has not been tested here.

[2] Dataset: http://www.cvl.isy.liu.se/research/datasets/stacking-dataset/

Figure 6: Results for Table (iPhone 4s), Grass (iPhone 5), and Books (iPhone 5) datasets. Left: Full frames after stacking with our method. Right columns: first frame in sequence, stacking of original frames, denoised first frame using Photoshop and our results. (Best viewed electronically)

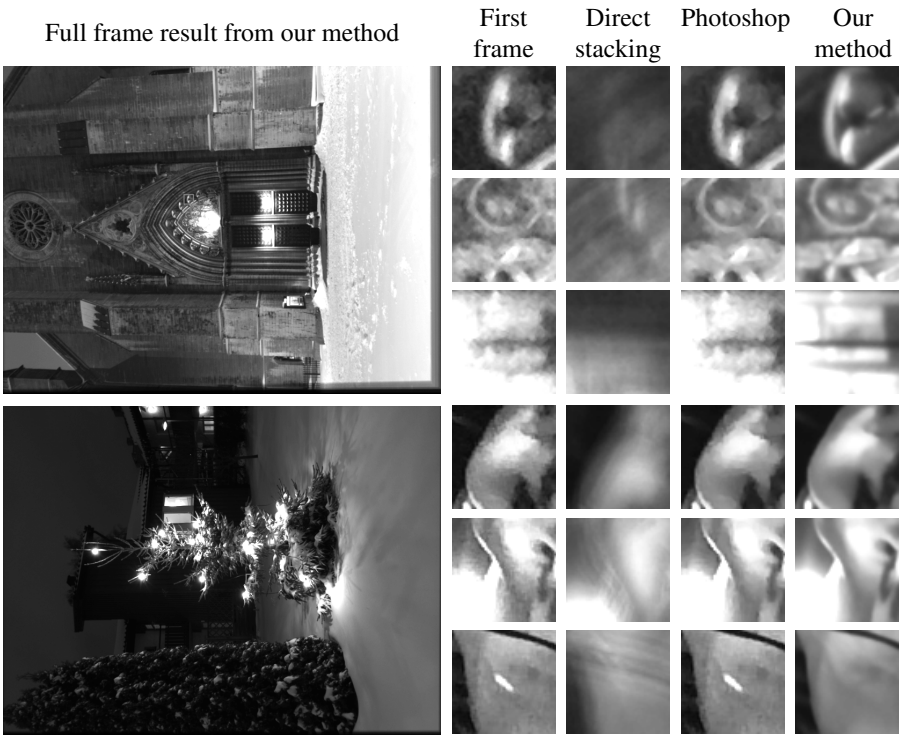


Figure 7: Results for Church (iPhone 4s) and Tree (iPhone 5s) datasets. Left: Full frame after stacking with our method. Right columns: first frame in sequence, stacking of original frames, denoised first frame using Photoshop and our results. (Best viewed electronically)

5.2 Comparative Experiments

Here we compare our results with a single frame from the stack, a denoising of this frame using Photoshop CS 5.1, and the average of the non-aligned frames in the sequence, see Figures 6 and 7. The Photoshop denoising was set to standard settings except "strength" and "preserve details" which were changed to 10 and 10% respectively.

It is also interesting to see how our method compares to a global alignment of the frames. In order to do this we used our estimated motion and applied a global rotation and translation on each frame. Please note that this motion has been estimated taking rolling shutter into account and that the first row of the global alignment will thus be the same as in our method. Figure 3 shows how the global alignment gets worse further down the image, whereas the proposed method has consistent performance at all image rows. As can be seen in the detail subplots, our algorithm successfully averages out the noise, while preserving structural details. An interesting observation is that our algorithm often manages to average out lens flares, (see e.g. Figure 7, bottom), as these

move around quite a bit when shooting handheld photos. In the case of lens-flares this is a desired behaviour, but of course actual scene objects that move during the $1-2$ sec exposure will also be averaged out.

5.3 Limitations

The proposed method, in essence, implements the behaviour of a tripod, and as such the final result is sensitive to moving objects in the scene. For small objects, the translation estimation will lock on to the background scene, and the moving objects will be smeared just like on a tripod. For large objects, on the other hand, the translation estimation will tend to lock on to the object instead. If the object satisfies the assumption of a fronto-parallel plane, see section 2.2, the result will be a sharp object, and a smeared background. In general however the result is unpredictable. Note however that other stacking functions than the frame average that we currently use, may to some extent remedy these limitations.

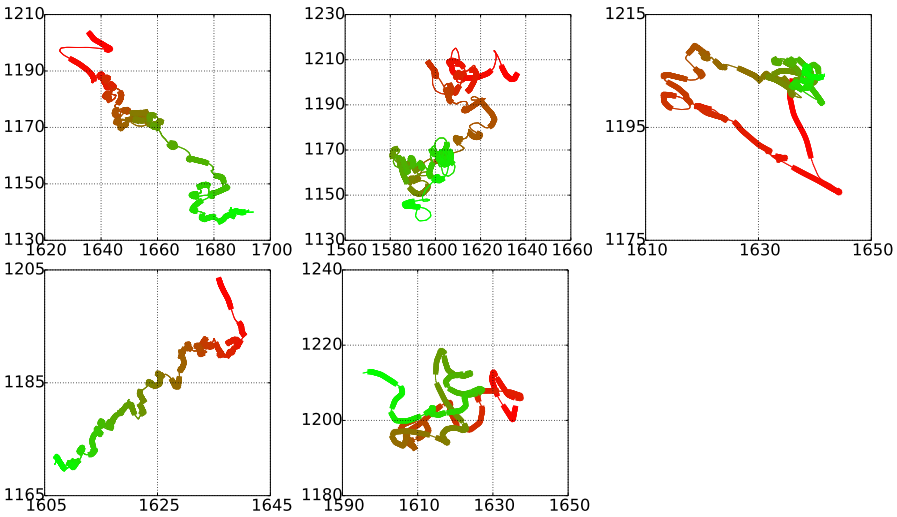


Figure 8: Trace of central pixel for the five stacks used in our experiments. Colours indicate time, ranging from red to green, and thick segments indicate the 32 individual exposures. Left to right, top to bottom: Table (4s), Grass (5), Books (5), Church (4s), Tree (5s)

5.4 Estimated PSFs from Gyroscope data

We have used an externally mounted L3G4200D gyro to record the device motion at 800Hz, as illustrated in Figure 1. Using the built-in gyro recording at 100Hz in the three iPhones (4s,5,5s), we obtain similar pixel traces, shown in Figure 8. By comparing the curves in Figure 1 and Figure 8, we see that the curves are similar, and thus conclude that the 100Hz sampling is sufficient.

In Figure 9 we have plotted iso-contours of the effective PSF for the central pixel in each of the datasets. The contour levels are set to $1/255$, $2/255 \ldots 32/255$. This means that beyond the first iso contour, the central pixel will not be influenced beyond the 8-bit quantization level. Beyond the second contour, a change of more than 128 in pixel value is required to influence the blurred pixel value, and so on.

It is interesting to relate the PSFs in Figure 9 to the imaging situation. The Grass sequence was imaged with elbows resting on a ledge, and consequently it has the smallest PSF. The Church and Tree sequences were recorded in cold weather, and consequently they have slightly more handshake. The Table, Books, and the 800Hz recording in Figure 1 were all recorded in warmer conditions, and consequently have better concentrated PSFs.

5.5 Quantitative Experiments

For quantitative evaluation of the stacking result, we use the standard deviation in time across a stack of frames. This measure is then averaged across all pixels to obtain a

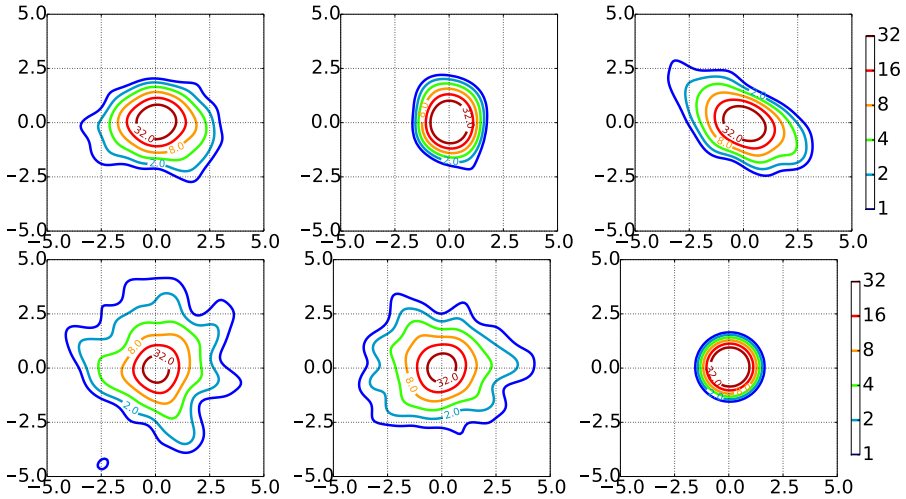


Figure 9: Iso contours of the effective PSF for central pixel. PSF values are scaled by 255, see text for details. Left to right, top to bottom: Table (4s), Grass (5), Books (5), Church (4s), Tree (5s), Gaussian with $\sigma = 0.5$.

scalar measure. If we denote the c-th colour band of RGB frame k in a stack by $I_{k,c}(\mathbf{x})$, the measure is computed as:

$$\sigma_{\text{avg}} = \frac{1}{3|\Omega|} \sum_{\mathbf{x}\in\Omega} \sum_{c=1}^{3} \sqrt{\frac{1}{K} \sum_{k=1}^{K} (I_{k,c}(\mathbf{x}) - I_{\text{avg},c}(\mathbf{x}))^2}\,, \tag{14}$$

$$\text{where} \quad I_{\text{avg},c}(\mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} I_{k,c}(\mathbf{x})\,. \tag{15}$$

Here Ω is the set of image coordinates in the frames, and $|\Omega|$ is the set size.

In Table 1 we use the measure (14) to compare different stacking approaches. Here **Global** refers to the global-shutter based frame alignment, also used in Figure 3. The other methods (**Slerp**, **Cubic**, **Quartic**, and **B-spline**) are versions of our rolling-shutter based algorithm, with different interpolation kernels. As can be seen in Table 1, all rolling-shutter based stacking approaches are significantly better than the global-shutter based alignment. We can also see that the **B-spline** kernel is slightly better than the other approaches. The reason for this is probably that its low-pass characteristic results in a denoising of the gyro signal. It is also interesting to note that **Slerp** performs surprisingly well. This may be caused by it being linear, just like the trapezoid integration in (13).

We have also tried blending the **Quartic** and the **B-spline** kernels according to (12), and then got the best results for a pure **B-spline** kernel. As the performance differences are quite small, these results may however not be significant, and are thus excluded from the Table.

Dataset iPhone Device		Table 4s	Grass 5	Books 5	Church 4s	Tree 5s
Global	σ_{avg}	6.17	6.83	7.55	8.02	5.23
Slerp	σ_{avg}	6.00	4.80	6.43	7.01	**4.31**
	residual	0.707	**0.611**	1.11	1.10	1.04
Cubic	σ_{avg}	6.00	4.77	6.42	7.02	4.31
	residual	0.728	0.612	1.12	1.16	1.11
Quartic	σ_{avg}	6.00	4.78	6.41	7.02	4.31
	residual	0.721	0.612	1.11	1.13	1.09
B-spline	σ_{avg}	**6.00**	**4.77**	**6.41**	**7.00**	4.31
	residual	**0.692**	0.612	**1.09**	**1.05**	**0.983**

Table 1: Quantitative results for different versions of our method, on the five datasets "Table", "Grass", "Books", "Church", and "Tree". The measure σ_{avg} is defined in (14), and "residual" is the mean squared residual of the reprojection error on tracked features. Best results in each column are shown in boldface. See Figures 6 and 7 for images of the different datasets.

6 Concluding Remarks

We have introduced an algorithm for accurate stacking of full resolution frames on a smartphone. This algorithm enables hand-held capture of low-noise images in low-light conditions, and thus implements a virtual tripod. This is accomplished using high accuracy motion estimation using logged gyro sensor data and correspondences from image feature tracking. In the experiments we demonstrate that the use of cumulative quaternion splines for motion interpolation results in a more accurate stacking than currently used stacking approaches that implicitly assume a global shutter. We also demonstrate that, while Photoshop style denoising works well in some situations, our algorithm consistently delivers a sharp, low-noise output.

The proposed motion estimation could also be used in other applications where accurate frame alignment is needed, such as flash-no-flash photography, and HDR imaging using exposure brackets.

When stacking is used in astrophotography, it is common to apply deconvolution to the stacking result, e.g. using Richardson-Lucy (RL) [16, 13]. This could also be done here to obtain a sharper final image. Since the PSFs in each frame are different, another possibility is to apply RL to the individual frames, *before* stacking. This will come at a higher computational cost, but as ringing artifacts tend to depend on both image structure and image smear, this may improve the output quality.

In the paper we have only investigated frame combination using direct averaging. In future research we plan to investigate how this compares to other commonly used approaches, such as temporal median and bilateral filtering. It would also be interesting to investigate criteria for stopping frame acquisition automatically when sufficient data is available to average out the noise. Finally, moving the entire algorithm onto a smartphone will also be tested.

References

[1] K. Brasch. The origin of stacking. *Sky and Telescope*, March 2014.

[2] Sony RX100 review. www.dpreview.com, accessed December 18, 2013.

[3] E. Eisemann and F. Durand. Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph.*, 23(3):673–678, Aug. 2004.

[4] P.-E. Forssén and E. Ringaby. Rectifying rolling shutter video from hand-held devices. In *CVPR10*, San Francisco, USA, June 2010. IEEE Computer Society.

[5] M. Grundmann, V. Kwatra, D. Castro, and I. Essa. Effective calibration free rolling shutter removal. *IEEE ICCP*, 2012.

[6] G. Hanning, N. Forslöw, P.-E. Forssén, E. Ringaby, D. Törnqvist, and J. Callmer. Stabilizing cell phone video using inertial measurement sensors. In *2nd IEEE IWMV*, 2011.

[7] B. K. P. Horn. Solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am.*, 4:629–642, April 1987.

[8] N. Joshi, S.-B. Kang, L. Zitnick, and R. Szeliski. Image deblurring using inertial measurement sensors. In *SIGGRAPH'10*, 2010.

[9] A. Karpenko, D. Jacobs, J. Baek, and M. Levoy. Digital video stabilization and rolling shutter correction using gyroscopes. Technical Report CSTR 2011-03, Stanford University Computer Science, September 2011.

[10] M.-J. Kim, M.-S. Kim, and S. Y. Shin. A general construction scheme for unit quaternion curves with simple high order derivatives. In *SIGGRAPH'95*, pages 369–376, 1995.

[11] C. Liu and W. Freeman. A high-quality video denoising algorithm based on reliable motion estimation. In *European Conference on Computer Vision (ECCV10)*, 2010.

[12] S. Lovegrove, A. Patron-Perez, and G. Sibley. Spline fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras. In *British Machine Vision Conference (BMVC'13)*, 2013.

[13] L. B. Lucy. An iterative technique for the rectification of observed distributions. *Astron. J.*, 79:745–754, 1974.

[14] G. Petschnigg, M. Agrawala, H. Hoppe, R. Szeliski, M. Cohen, and K. Toyama. Digital photography with flash and no-flash image pairs. In *Proceedings of SIGGRAPH'04*, 2004.

[15] T. Portz, L. Zhang, and H. Jiang. High-quality video denoising for motion-based exposure control. In *2nd IWMV*, 2011.

[16] W. H. Richardson. Bayesian-based iterative method of image restoration. *J. Opt. Soc. Am.*, pages 55–59, 1972.

[17] E. Ringaby and P.-E. Forssén. Efficient video rectification and stabilisation for cell-phones. *IJCV*, 96(3), 2012.

[18] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer Verlag, 2011.

[19] O. Šindelář and F. Šroubek. Image deblurring in smartphone devices using built-in inertial measurement sensors. *Journal of Electronic Imaging*, 22(1), 2013.

[20] J. Weickert. *Anisotropic Diffusion in Image Processing*. ECMI Series, Teubner Verlag, Stuttgart, Germany, 1998.

[21] O. Whyte, J. Sivic, A. Zisserman, and J. Ponce. Non-uniform deblurring for shaken images. In *IEEE CVPR*. IEEE Computer Society, June 2010.

[22] L. Yuan, J. Sun, L. Quan, and H.-Y. Shum. Image deblurring with blurred/noisy image pairs. In *SIGGRAPH'07*, 2007.

[23] H. Zhang and D. Wifp. Non-uniform camera shake removal using a spatially-adaptive sparse penalty. In *NIPS13*, 2013.