The Dissertation Committee for Chao Jia
certifies that this is the approved version of the following dissertation:

Video Stabilization and Rectification for Handheld Cameras

Committee:

Brian L. Evans, Supervisor

Alan Bovik

Constantine Caramanis

Donald Fussell

Todd Humphreys

Video Stabilization and Rectification for Handheld Cameras

by

Chao Jia, B.E.; M.S.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2014

Dedicated to my wife Fang and my parents.

Acknowledgments

I would like to first express my deepest gratitude to Prof. Brian Evans for being an excellent advisor and mentor. I am grateful not only for his insightful guidance and support throughout my graduate studies, but also for his integrity, discipline, and attention to details, which have all influenced my life. His genuine dedication to research and teaching is exceptional. I feel extremely lucky to be his student.

I would like to thank my thesis committee members, Prof. Alan Bovik, Prof. Constantine Caramanis, Prof. Donald Fussell, and Prof. Todd Humphreys, for their thoughtful comments and suggestions that strengthened this thesis. I am also grateful to Prof. Kristen Grauman, Prof. Sujay Sanghavi, and Prof. David Morton. I have learned a lot from their great courses in computer vision, machine learning and optimization.

Thanks to Dr. Hamid Sheikh for introducing me to the problem of video stabilization for rolling shutter cameras. Thanks to Dr. Xiaolong Huang and Dr. Ramin Rezaiifar for mentoring my internships at Qualcomm Research.

My labmates in ESPL deserve much thanks for making my graduate life memorable. Thank you Greg Allen, Hugo Andrade, Aditya Chopra, Marcus DeYoung, Kapil Gulati, Debarati Kundu, Jing Lin, Yousof Mortazavi, Marcel Nassar, Karl Nieman, Hamood Rehman, Zeina Sinno and Kyle Wesson.

Special thanks to Kapil for your kind help in writing my first paper.

Finally, this thesis would not have been possible without the love and support from my family. I am deeply grateful to my father Qiuming Jia, my mother Wenhui Yu and my wife Fang Lu. I dedicate this thesis to them.

Video Stabilization and Rectification for Handheld Cameras

Publication No. _____

Chao Jia, Ph.D.
The University of Texas at Austin, 2014

Supervisor: Brian L. Evans

Video data has increased dramatically in recent years due to the prevalence of handheld cameras. Such videos, however, are usually shakier compared to videos shot by tripod-mounted cameras or cameras with mechanical stabilizers. In addition, most handheld cameras use CMOS sensors. In a CMOS sensor camera, different rows in a frame are read/reset sequentially from top to bottom. When there is fast relative motion between the scene and the video camera, a frame can be distorted because each row was captured under a different 3D-to-2D projection. This kind of distortion is known as rolling shutter effect. Digital video stabilization and rolling shutter rectification seek to remove the unwanted frame-to-frame jitter and rolling shutter effect, in order to generate visually stable and pleasant videos. In general, we need to (1) estimate the camera motion, (2) regenerate camera motion, and (3) synthesize new frames. This dissertation aims at improving the first two steps of video stabilization and rolling shutter rectification.

It has been shown that the inertial sensors in handheld devices can provide more accurate and robust motion estimation compared to vision-based methods. This dissertation proposes an online camera-gyroscope calibration method for sensor fusion while a user is capturing video. The proposed method uses an implicit extended Kalman filter and is based on multiple-view geometry in a rolling shutter camera model. It is able to estimate the needed calibration parameters online with all kinds of camera motion.

Given the camera motion estimated from inertial sensors after the proposed calibration method, this dissertation first proposes an offline motion smoothing algorithm based on a 3D rotational camera motion model. The offline motion smoothing is formulated as a geodesic-convex regression problem on the manifold of rotation matrix sequences. The formulated problem is solved by an efficient two-metric projection algorithm on the manifold. The geodesic-distance-based smoothness metric better exploits the manifold structure of sequences of rotation matrices. Then this dissertation proposes two online motion smoothing algorithms that are also based on a 3D rotational camera motion model. The first algorithm extends IIR filtering from Euclidean space to the nonlinear manifold of 3D rotation matrices. The second algorithm uses unscented Kalman filtering on a constant angular velocity model. Both offline and online motion smoothing algorithms are constrained to guarantee that no black borders intrude into the stabilized frames.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Handheld video cameras, especially in cell phones, have become increasingly popular today because of their portability and price. With handheld cameras, consumers are able to shoot and share videos at anytime and anywhere conveniently. However, the quality of videos shot by handheld cameras is severely affected by unintentional camera motion, such as the up-and-down motion caused by walking or simply jitter caused by hand shake. In addition, rolling shutter effect often exists when there is fast camera motion, whether it is intentional or not.

Rolling shutter effect [23, 26] is a common kind of distortion on CMOS image sensors. CMOS image sensors dominated the cellphone camera sensor market over CCD sensors due to lower power consumption and faster data throughput. In a CMOS sensor camera, usually different rows in a frame are read/reset sequentially from top to bottom, as shown in Fig. 1.2. When there is fast relative motion between the scene and the video camera, a frame can be distorted because each row was captured under a different 3D-to-2D projection. The rolling shutter effect usually includes skew, smear and wobble distortion [4]. Fig. 1.3 shows an example of the skew distortion caused by a

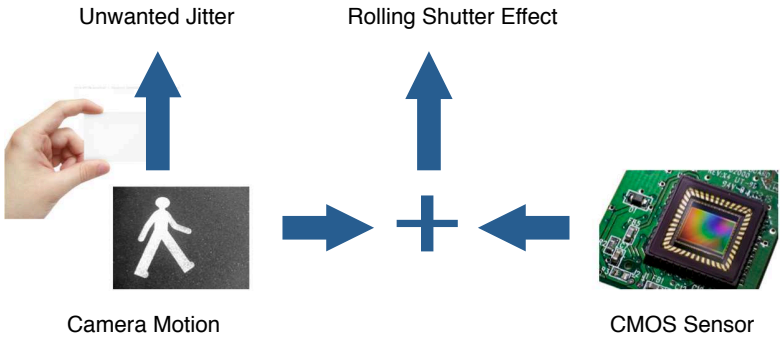


Figure 1.1: Unwanted jitter and rolling shutter effect caused by camera motion and CMOS sensor.

rolling shutter camera, and the rectified frame.

Please note that not all CMOS image sensors utilize a rolling shutter. Moreover, a rolling shutter is not only used by CMOS image sensors. In fact, rolling shutter has a long history and has been used to record almost every film over the last century. Besides, rolling shutter effect can sometimes also be found in augmented reality rendering. In this dissertation, however, I only focus on the rolling shutter effect brought by CMOS image sensors in handheld cameras.

This dissertation aims at rectifying the rolling shutter effect and removing the unwanted jitter in videos (also known as "video stabilization"). Both rolling shutter effect rectification and video stabilization consist of three major steps [62]: (1) camera motion estimation, (2) camera motion regeneration and (3) frame synthesis (as shown in Fig. 1.4).

In the first step, only one camera pose is needed for each frame if the

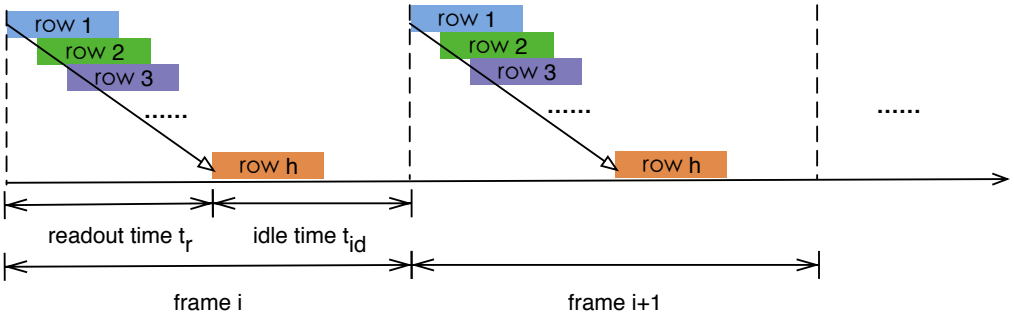


Figure 1.2: Rows are captured sequentially in rolling shutter cameras. Each block represents the exposure time of a certain row.

Figure 1.3: An example of a frame with rolling shutter effects (left) and the rectified frame (right). The skew distortion is caused by fast panning motion of the camera.

video is captured by a global shutter camera. However, for rolling shutter cameras, we have to estimate camera motion for each row.

In the second step, rolling shutter effect rectification just needs to fix a unique camera motion for all of the rows in each frame, while video stabilization needs to smooth the sequence of camera motions of all of the frames. Actually one can understand rolling shutter rectification as an intra-frame video stabilization.

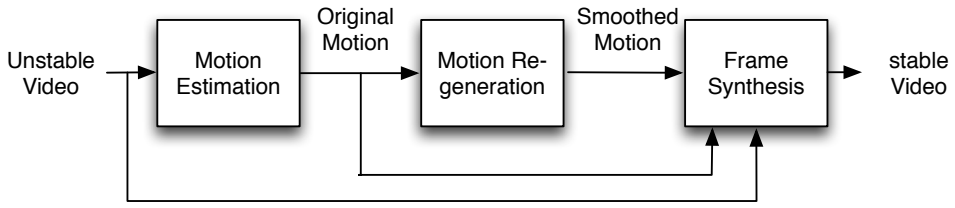


Figure 1.4: Three major steps for video stabilization (same for video rectification).

In the last step the new frames are synthesized based on the difference between the original and the regenerated camera motion.

1.1 Motion Estimation and Sensor Calibration

1.1.1 Camera Motion Model and Motion Estimation

Traditional camera motion estimation assumes a global camera pose for each frame, and is thus not suitable for videos with rolling shutter effect. To estimate the camera motion, we first need to choose a proper camera motion model. Many (if not most) existing camera motion methods use 2D models, including translational model, similarity model, affine model, etc. These 2D models are simple but fundamentally limited to represent the real 3D camera motion. Compared to 2D motion models, 3D motion models can accurately reflect the real camera perspective projection, and thus make more sense and are able to avoid image distortion in the frame synthesis step [56].

A full 3D motion model consists of both 3D rotation and 3D translation of the camera. Compared to rotation, estimating 3D camera translation is

a very difficult problem. In addition, the frame synthesis step considering 3D camera translation needs the depth value for every pixel, which is very difficult to obtain unless light field camera is used [80]. Therefore, many existing approaches ignore the 3D translation of camera and only consider the 3D rotation. This simplification in the 3D model is reasonable since both rolling shutter effect and the unwanted jitter in videos are primarily caused by camera rotation [23, 46].

Many camera motion estimation methods rely only on the visual information (either intensity values of all the pixels in the frame or locations of the the tracked feature points that appeared in the video frames). This kind of methods are computationally intensive and prone to errors in the visual information. Especially for a rolling shutter camera, vision-based methods cannot reliably obtain an accurate camera motion estimate at a row-based resolution.

For this reason, inertial sensors such as gyroscopes and accelerometers, which can be found in many modern smartphones and tablets, have been used to help estimate camera motion because of their increasing accuracy, high sampling rate and robustness to lighting conditions. In some video stabilization algorithms, camera motion is estimated directly using the inertial sensors without the help of video sequences [31, 46]. The estimation time is thus greatly shortened, which makes real-time rectification possible.

1.1.2 Calibration and Synchronization of Camera and Gyroscope

In the field of robotics and navigation, fusion of visual and inertial information has long been used to provide more accurate and reliable camera motion tracking [67, 84]. The fusion of camera and inertial sensors, however, requires precise calibration: the coordinate system of inertial sensors does not coincide with that of camera, and the timestamps of inertial sensor readings and video frames are not well synchronized. Apart from the relative pose and timestamp delay, camera and inertial sensors themselves also have to be calibrated so that essential parameters such as focal length and sensor biases are known. The rolling shutter camera model adds even more parameters to the calibration problem.

Many existing approaches in visual-inertial sensor fusion assume that calibration and synchronization have been done offline beforehand. Moreover, camera self-calibration (estimation of camera intrinsic parameters) is usually executed separately from relative pose and delay calibration between camera and inertial sensors. Some calibration methods can be only performed in laboratory environments with special devices (e.g. spin table and checkerboard), which further prevents everyday users from using cellphone cameras conveniently with the help of inertial sensors. It would be better if cellphone cameras and inertial sensors could be calibrated and synchronized online while users capture videos, without any prior knowledge about the devices or any special calibration hardware.

Sometimes we do not have to calibrate both gyroscope and accelerom-

eter. For instance, if we only care about 3D rotation instead of both rotation and translation, then gyroscope is the only inertial sensor we need to calibrate and synchronize with the cellphone camera.

1.2 Motion Re-generation

The camera motion re-generation step comes after camera motion estimation. As aforementioned, to rectify the rolling shutter effect we only need to fix a unique camera pose for all the rows in the same frame. The more difficult part is to smooth the camera motion sequence in order to stabilize the video sequence.

Due to the camera motion change from motion smoothing, some areas in the synthesized frame will be undefined. This is known as black border problem, as shown in Fig. 1.5. In practice we have to crop the resulting video frames and enlarge them if necessary. Still, in motion smoothing, we have to constrain the change of camera motion in order to guarantee that no black borders intrude into the stabilized video frames. How to take such constraint into consideration optimally is a challenging problem.

Like motion estimation, motion smoothing methods still depend on the choice of camera motion model. Motion smoothing using 2D models are based on distances defined in the Euclidean space. Under 3D rotational model, the camera motion for a video can be considered as a sequence of 3D rotation matrices. The 3D rotation matrices lie on a non-linear manifold $\mathbf{SO(3)}$. Thus the geodesic distance defined on this manifold is a more natural choice than

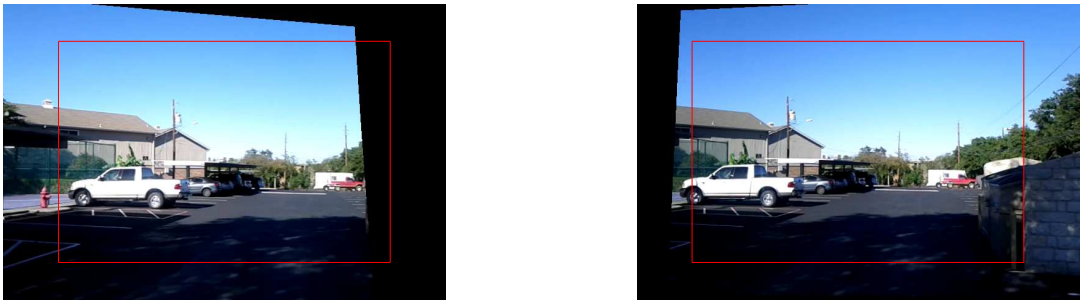


Figure 1.5: Illustration of the black border problem. Black border intrudes into the cropping window in the left image, but not in the right image.

the Euclidean distance defined on the Euclidean space that this manifold is embedded on.

Given the choice of motion model, motion smoothing algorithms fall into two categories: offline smoothing and online (real-time) smoothing.

1.2.1 Offline Motion Smoothing

Offline smoothing smooths the camera motion sequence after the entire video is captured. Many (if not most) existing motion smoothing algorithms are offline smoothing on 2D motion models. Gaussian window filtering was used to smooth the entire camera motion path in [19, 62] under 2D translational and affine model respectively. Another kind of algorithms smooths the camera motion via minimizing a certain objective function that represents the smoothness of the camera motion trajectory. An advantage of such objective-minimizing methods is that the black-border constraints can be naturally added to the problem and solved by constrained optimization methods.

In [82], the authors defined the objective function as the L^2 norm of the second order difference of camera motion under 2D Euclidean model. The black border constraint was approximately modeled by an interval constraint on the motion parameters. Similar modeling was also used in [73], but the variables were assumed to be integer-valued and the problem was solved via dynamic programming. In [29] the objective function was a mixture of the first, second and third order difference of camera motion measured by the L^1 norm. The motion model was 2D similarity motion and the black-border constraint was modeled precisely as linear inequalities. As a result, the constrained motion smoothing could be solved efficiently by linear programming.

Given the geodesic distance defined on $\mathbf{SO(3)}$, 3D rotation smoothing has been implemented locally by low-pass filtering [31, 56]. However, so far we have not found any previous work that smooth the 3D camera rotation sequence globally based on the geodesic distance. Global motion smoothing has been shown superior to local smoothing in 2D camera motion models [29] and therefore is tempting to try it for the 3D rotational motion model.

1.2.2 Online (Real-Time) Motion Smoothing

While most existing approaches address motion smoothing as an offline processing after the entire video sequence has been recorded, online (real-time) video stabilization is necessary for real-time applications such as video conferencing and broadcasting. Besides, for consumers who just want to record videos, real-time stabilization can greatly improve the user experience with the

stabilized videos displayed in real-time on the viewfinders. Real-time video stabilization is also able to reduce the memory requirements with frames stabilized before compression. In real-time video stabilization, camera motion is required to be smoothed in a causal way. This is more difficult than offline motion smoothing because we are missing information of how camera motion changes afterward.

Existing work for real-time motion smoothing was restricted to 2D motion models. In [17] IIR filtering was proposed for online motion smoothing based on 2D translational motion model. Kalman filtering was first used for online smoothing in [18]. The intentional motion parameters (under 2D translational motion model) were modeled by a constant velocity linear system so Kalman filtering could be used to optimally estimate them. The same Kalman-filtering motion smoothing framework was extended to 2D affine motion model in [55], leading a better performance. The same algorithm was widely used in the later video stabilization works, such as [92].

The black-border constraints were rarely considered in online motion smoothing. In [86] the authors proposed to use constrained Kalman filtering for 2D translational motion model. Because of the simplicity of the motion model, interval constraints could be used and the constrained estimate could be obtained in one step.

1.3 Frame Synthesis

Frame synthesis is the last step of rolling shutter effect rectification and video stabilization. In this step we need to generate the new frames based on the difference between the original camera motion and the computed camera motion in step (2). For 2D or 3D pure rotational motion model, frame synthesis can be efficiently implemented by 2D image transformations (3D rotation change corresponds to 2D homographic transformation).

A problem is that the generated locations of the original pixels may not have integer values and are thus not exactly on the vertices of the pixel grid in the new frame. In practice inverse warping with a certain interpolation method (usually bilinear interpolation) is used to solve this problem. However, the homographic transformation to rectify the rolling shutter effect is not invertible as that for video stabilization. Therefore, forward warping has to be used if we need to precisely correct the rolling shutter effects. Forward warping is usually slower than inverse warping, but GPUs can efficiently handle it in real time using texture mapping technology [33].

1.4 Dissertation Summary

In this dissertation, I focus on the first two steps of video stabilization and rolling shutter rectification. For motion estimation, I develop an online algorithm for camera and gyroscope calibration without any prior knowledge of the devices. The camera motion can be reliably obtained from gyroscope after the calibration and synchronization is done. For motion smoothing, I first pro-

pose an offline motion smoothing algorithm based on a 3D rotational camera motion model. The offline motion smoothing is formulated as a geodesic-convex regression problem on the manifold of rotation matrix sequences. The formulated problem is solved by an efficient two-metric projection algorithm on the manifold. The geodesic-distance-based smoothness metric better exploits the manifold structure of sequences of rotation matrices. Then I propose two online motion smoothing algorithms that are also based on a 3D rotational camera motion model. The first algorithm extends IIR filtering from Euclidean space to the nonlinear manifold of 3D rotation matrices. The second algorithm uses unscented Kalman filtering on a constant angular velocity model. Both offline and online motion smoothing algorithms are constrained to guarantee that no black borders intrude into the stabilized frames.

1.4.1 Thesis Statement

In this dissertation, I defend the following thesis statement:

For handheld cameras with CMOS sensors, videos can be satisfactorily rectified and then stabilized either online or offline, with the camera motion estimated directly from gyroscopes after effective sensor calibration.

1.4.2 Summary of Contributions

The main contributions of this dissertation can be summarized as follows.

1. **Online Camera-Gyroscope Auto-Calibration for Cellphones**: In

this contribution, I develop an online method that estimates all of the necessary camera and gyroscope parameters while a user is capturing video. This algorithm is based on an implicit extended Kalman filter (EKF). Each video frame provides a view of the 3D scene and triggers the update of the EKF through multiple-view geometry. By extending the recent proposed multiple-view coplanarity constraint of camera rotation [49] to rolling shutter cameras, I propose a novel implicit measurement that involves only camera rotation but works for any camera translation, including zero translation (pure rotation). The implicit measurements can be effectively used in the EKF to update the estimate of state vectors. This algorithm is able to estimate the needed calibration and synchronization parameters online with all kinds of camera motion, and can be embedded in video stabilization for fast camera motion estimation using gyroscopes. Both Monte Carlo simulation and cellphone experiments show that this online calibration and synchronization method converges fast to the ground truth values.

2. **Constrained 3D Rotation Smoothing via Global Manifold Regression**: In this contribution, I present a novel offline motion smoothing algorithm for video stabilization. I use a pure 3D rotation motion model with known camera projection parameters. I directly smooth the sequence of camera rotation matrices for the video frames by exploiting the Riemannian geometry on a manifold. I consider the entire set of sequences of rotation matrices as a Riemannian manifold. This al-

lows me to formulate the offline motion smoothing problem globally as a regression problem on the manifold based on geodesic distance. I introduce a geodesic-convex constraint on the manifold to approximate black-border constraint so that the entire motion smoothing problem is kept geodesic-convex on the manifold. To solve the formulated constrained smoothing problem on the manifold, I compute the gradient and Hessian of the objective function using Riemannian geometry, and then extend the two-metric projection algorithm in Euclidean space to non-linear manifolds.

The geodesic-distance-based smoothness metric better exploits the manifold structure of sequences of rotation matrices. The geodesic-convex constraints effectively guarantee that no black borders intrude into the stabilized frames. The proposed manifold optimization algorithm can find the global optimal solution in only a few iterations. Experimental results show that my motion smoothing method outperforms state-of-the-art methods by generating more stable videos with less distortion.

3. **Real-time 3D Rotation Smoothing**: In this contribution, I propose two real-time motion smoothing algorithms for video stabilization using a pure 3D rotation motion model with known camera projection parameters. Both proposed algorithms aim at smoothing 3D rotation matrix sequences in a causal way. The first algorithm smooths the 3D rotation sequences in a way similar to 1st-order IIR filtering. The second algorithm uses sequential probabilistic estimation under a constant angular

velocity model. These two algorithms are generalized from classical 2D motion smoothing algorithms. I exploit the manifold structure of the rotation matrices so that the proposed algorithms directly smooth the 3D rotation sequences on the manifold. In addition, I introduce a simple projection step in order to guarantee that no black borders intrude into the stabilized video frames. Experimental results show that the proposed algorithms are able to effectively stabilize video sequences and outperform their 2D counterparts with less jitter and distortion.

1.5 Organization

This dissertation is organized as follows:

Chapter 2 presents an online camera self-calibration and camera-gyroscope calibration algorithm for handheld cameras with rolling shutter. The calibration algorithm is able to estimate all the necessary parameters so that after calibration the 3D camera rotation for every video frame can be obtained directly from gyroscope readings. The subsequent chapters assume that camera motion has been estimated using gyroscope.

Chapter 3 proposes to formulate the offline motion smoothing problem as a constrained manifold regression on the manifold of rotation matrix sequences. An extension of two-metric scaled projection method is proposed to solve such problem efficiently.

Chapter 4 proposes the IIR-like and UKF-based algorithms for online

3D rotation smoothing. Ad-hoc estimate projection is used to guarantee no black borders intrude the stabilized frames.

Chapter 5 concludes the dissertation by summarizing the contributions and provides suggestions for future work.

1.6 List of Acronyms

CCD Charge-Coupled Device

CMOS Complementary Metal-Oxide-Semiconductor

CV Constant Velocity

EKF Extended Kalman Filter

FPS Frame per Second

GPB Generalized Pseudo-Bayesian

GPU Graphics Processing Units

IIR Infinite Impulse Response

KF Kalman Filter

KLT Kanade-Lucas-Tomasi

MAP Maximum a Posteriori

MVI Motion Vector Integration

PDF Probability Density Function

PSD Positive Semi-Definite

QP Quadratic Programming

RMS Root Mean Square

SLAM Simultaneous Localization and Mapping

SO(3) 3-dimensional Orthogonal Group

SOG Sum of Gaussian

UKF Unscented Kalman Filter

UT Unscented Transform

Chapter 2

Online Camera-Gyroscope Auto-Calibration for Cellphones

2.1 Introduction

Cellphone cameras have been increasingly popular for video capture due to their portability and processing power of cellphones. An increasing number of users are getting accustomed to recording their memorable events using cellphone cameras. Beyond the video recording itself, video acquisition also provides opportunities for applications such as augmented reality and visual odometry. No matter what application mobile video capture is used for, camera motion estimation is an essential step to improve the video quality and better analyze the video content.

Handheld mobile devices such as cellphones usually suffer from ego-motion that is changing very fast, which makes it difficult to track the camera accurately using only the captured videos. For this reason, inertial sensors on cellphones such as gyroscopes and accelerometers have been used to help estimate camera motion because of their increasing accuracy, high sampling rate and robustness to lighting conditions. It has been shown that through the fusion of visual and inertial information, camera motion can be estimated

more accurately and reliably [67, 84]. The fusion of camera and inertial sensors, however, requires precise calibration: the coordinate system of inertial sensors does not coincide with that of camera, and the timestamps of inertial sensor readings and video frames are not well synchronized. Apart from the relative pose and timestamp delay, camera and inertial sensors themselves also have to be calibrated so that essential parameters such as focal length and sensor biases are known.

Many existing approaches in visual-inertial sensor fusion assume that calibration and synchronization have been done offline beforehand. Moreover, camera self-calibration (estimation of camera intrinsic parameters) are usually executed separately from relative pose and delay calibration between camera and inertial sensors [11, 83]. Some calibration methods can be only performed in laboratory environments with special devices (e.g. spin table and checkerboard) [58, 93], which further prevents everyday users from using cellphone cameras conveniently with the help of inertial sensors. In this chapter, I focus on online calibration and synchronization of cellphone cameras and inertial sensors while users capture videos, without any prior knowledge about the devices or any special calibration hardware.

Unlike traditional cameras, most cellphone cameras do not capture the rows in a single frame simultaneously, but sequentially from top to bottom. When there is fast relative motion between the scene and the camera, a frame can be distorted because each row was captured under different 3D-to-2D projections. This is known as rolling shutter effect [4, 23, 26] and has to be

considered in calibration and fusion of visual and inertial sensors.

Although some applications such as visual odometry require estimation of both camera rotation and translation, estimating rotation using only the gyroscope has been used successfully in video stabilization [46] and feature tracking [36]. When the displacement of pixels between consecutive video frames is primarily caused by camera rotation, a gyroscope-only approach successfully stabilized video and removed rolling shutter effects [31, 46]. Similarly, gyroscope measurements were used to pre-warp the frames so that the search space of the Kanade-Lucas-Tomasi (KLT) [60, 76] feature tracker can be narrowed down to its convergence region [36]. In these proposed methods there is no need to use the accelerometer. Therefore, only the camera and the gyroscope need to be calibrated. In this chapter I focus on such camera-gyroscope calibration. Although we only care about camera rotation, the proposed approach does not assume that there is no camera translation.

The proposed online calibration and synchronization is based on an extended Kalman filter (EKF). Each video frame provides a view of the 3D scene and triggers the update of the EKF through multiple view geometry. Although we care about camera rotation only, I do not assume any degeneration in the motion of the camera. By extending the recent proposed multiple-view coplanarity constraint of camera rotation [49] to rolling shutter cameras, I propose a novel implicit measurement that involves only camera rotation but works for any camera translation, including zero translation (pure rotation). The implicit measurements can be effectively used in the EKF to update the

estimate of state vectors.

This chapter is organized as follows. Section 2.2 reviews previous algorithms on camera self-calibration, camera-inertial calibration, and camera-gyroscope calibration. Section 2.3 introduces the rolling shutter camera model and summarizes the parameters that we need to estimate in this chapter. Section 2.4 presents the coplanarity constraint on camera rotation in the rolling shutter camera model. This constraint is then used in implicit measurements by the proposed EKF-based online calibration and synchronization approach in Section 2.5. Section 2.6 shows and analyzes the results of Monte Carlo simulation and cellphone experiments using the proposed approach. Section 2.7 concludes the chapter.

2.2 Related work

Camera self-calibration has been extensively studied [32] for both global shutter camera [93] and rolling shutter camera [71], but previous work on online self-calibration is somewhat rare. In [10] full-parameter online camera self-calibration is first proposed in the framework of sequential Bayesian structure from motion using a sum of Gaussian (SOG) filter. Their work assumes a global shutter camera model and the motion of the camera has to contain large enough translation to make the structure from motion problem well-conditioned.

The inertial sensors (gyroscope and accelerometer) are widely used in camera motion estimation and simultaneous localization and mapping (SLAM)

together with visual measurements [11, 43, 83]. Especially for hand-held devices such as cellphone cameras, inertial-aided approaches appear more robust in camera tracking and SLAM when compared to purely vision-based approaches [12, 69]. For more details about SLAM can be found in [3, 15].

Relative pose between inertial sensors and camera has been successfully estimated offline with special hardware [58] or simply with a known calibration pattern [63]. Online camera-inertial calibration has also been implemented recently in the framework of SLAM or navigation [47] together with the estimation of inertial sensor biases. However, to the best of my knowledge all of the previous work assumes that the camera itself has been calibrated; i.e., the camera projection parameters are known. Moreover, rolling shutter effect was not taken into account in the fusion of inertial and visual sensors until very recently [38, 53, 59]. The timestamp delay between camera and inertial sensors was always assumed as known except for the recent work in [52] which estimates the timestamp delay online.

The SLAM framework for online calibration of camera and inertial sensors involves estimation of camera translation and 3D scene structure. In addition, camera translation estimation and accelerometer calibration require large enough camera translation to initialize absolute scale and speed estimate [48, 61]. Therefore, such methods are too complicated if we only care about camera rotation and just want to use gyroscope to estimate and track camera motion.

To calibrate the camera and gyroscope system, the method in [46] pro-

posed to quickly shake the camera while pointing at a far-away object (e.g., a building). Feature points between consecutive frames are matched and all parameters are estimated simultaneously by minimizing the homographic re-projection errors under a pure rotation model. The calibration in [36] is also based on homography transformation of matched feature points assuming pure rotation, except that different parameters are estimated separately first and then refined through non-linear optimization. However, as shown in [36], when the camera translation is not negligible relative to the distance of the feature points to the camera, such pure rotation model becomes less accurate and the calibration results will deviate from the ground truth. The proposed calibration method differs with [36, 46] not only in that it is online estimation, but also in that it does not assume zero translation at all. Therefore, the proposed calibration can be performed implicitly anytime and anywhere while the camera is recording video. This is especially convenient for amateur photographers who want to take stabilized videos with smartphone cameras.

2.3 Rolling shutter camera model and gyroscope

Points in the camera reference space are projected according to the pinhole camera model. Assuming the 3D point coordinates in the camera reference space are $[X_c, Y_c, Z_c]^{\mathrm{T}}$, their projection onto the image plane can be represented as

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} c_x + f\frac{X_c}{Z_c} \\ c_y + f\frac{Y_c}{Z_c} \end{bmatrix}, \tag{2.1}$$

where f is the focal length and c_x, c_y are the principal point coordinates. Here I assume that the camera projection skew is zero and the pixel aspect ratio is 1 as in [10], which is a reasonable assumption for today's cellphone cameras. Similarly, given the pixel coordinate $[u_x, u_y]^{\mathrm{T}}$, we can invert (2.1) to obtain the 3D coordinates of the corresponding feature point in the camera reference space up to an unknown scale as

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \lambda \begin{bmatrix} u_x - c_x \\ u_y - c_y \\ f \end{bmatrix}. \tag{2.2}$$

Based on (2.2), I further model the radial lens distortion of camera using two distortion coefficients as

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \lambda \begin{bmatrix} (1 + \kappa_1 r^2 + \kappa_2 r^4)(u_x - c_x) \\ (1 + \kappa_1 r^2 + \kappa_2 r^4)(u_y - c_y) \\ f \end{bmatrix}, \tag{2.3}$$

where

$$r = \sqrt{\left(\frac{u_x - c_x}{f}\right)^2 + \left(\frac{u_y - c_y}{f}\right)^2}. \tag{2.4}$$

In rolling shutter cameras, rows in each frame are exposed sequentially from top to bottom [26, 54], as shown in Fig. 2.1. In Fig. 2.1 each block represents the exposure of a certain row. The exposure duration of each row (represented by the length of each block) depends on the lighting conditions. In this chapter we ignore possible image blur and assume instantaneous exposure. Thus, the exposure moment of each row can be approximated as the left end of each block in Fig. 2.1. For an image pixel $\mathbf{u} = [u_x, u_y]^{\mathrm{T}}$ in frame i, the exposure time can be represented as $t(\mathbf{u}, i) = t_i + t_r \frac{u_y}{h}$, where t_i is the timestamp for

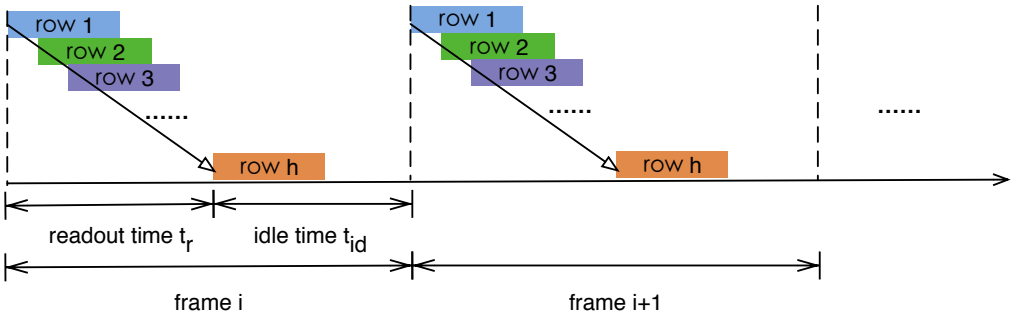


Figure 2.1: Rows are captured sequentially in rolling shutter cameras. Each block represents the exposure time of a certain row.

frame i and h is the total number of rows in each frame. Here t_r is the readout time for each frame, which is usually about $60\% - 90\%$ of the time interval between frames.

There exists a constant delay t_d between the recorded timestamps of gyroscope and videos. Thus using the timestamps of gyroscopes as reference, the exposure time of pixel $\mathbf{u}$ in frame i should be modified as

$$t(\mathbf{u}, i) = t_i + t_d + t_r \frac{u_y}{h}. \tag{2.5}$$

To use the gyroscope readings we also need to know $\mathbf{q}_c$, the relative orientation of the camera in the gyroscope frame of reference (represented in unit quaternion). Finally, the bias of the gyroscope $\mathbf{b}_g$ needs to be considered. Therefore, in the online calibration we need to estimate the parameters f, c_x, c_y, κ_1, κ_2, t_r, t_d, $\mathbf{b}_g$ and $\mathbf{q}_c$.

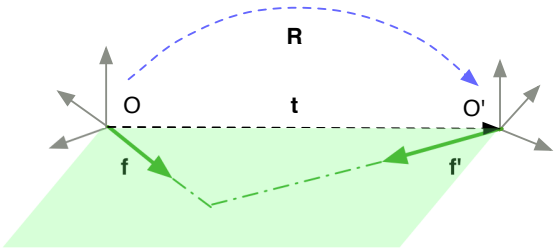


Figure 2.2: The epipolar constraint on a pair of features in two viewpoints.

2.4 Coplanarity constraint for camera rotation

The proposed calibration and synchronization rely on the constraints applied to camera rotations.

2.4.1 Coplanarity constraint in global shutter cameras

First let us consider a global shutter camera in which all of the pixels in the same frame are captured at the same time. Assume the normalized 3D coordinate vectors of a certain feature in two viewpoints (frames) are $\mathbf{f}_i$ and $\mathbf{f}'_i$ (note that by (2.3) we cannot recover the absolute scale but only the direction of the 3D feature vector). The well-known epipolar constraint [32] is

$$(\mathbf{f}_i \times \mathbf{R}\mathbf{f}'_i) \cdot \mathbf{t} = 0, \tag{2.6}$$

where $\mathbf{R}$ and $\mathbf{t}$ are the relative rotation and translation between the two viewpoints. The epipolar constraint means that the vectors $\mathbf{f}_i$, $\mathbf{R}\mathbf{f}'_i$ and $\mathbf{t}$ are coplanar, as shown in Fig. 2.2. Now assume that three or more features are observed in these two viewpoints. By the epipolar constraint all vectors $\mathbf{f}_i \times \mathbf{R}\mathbf{f}'_i$

are perpendicular to the relative translation vector $\mathbf{t}$, and thus are coplanar ($\mathbf{t}$ is the normal vector of such plane). Such coplanarity can be expressed by the determinant of the 3×3 matrix composed by any three $\mathbf{f}_i \times \mathbf{R}\mathbf{f}'_i$ vectors being zero

$$\det[(\mathbf{f}_1 \times \mathbf{R}\mathbf{f}'_1)|(\mathbf{f}_2 \times \mathbf{R}\mathbf{f}'_2)|(\mathbf{f}_3 \times \mathbf{R}\mathbf{f}'_3)] = 0. \tag{2.7}$$

This coplanarity was introduced in [49] and does not depend on the camera translation at all. Another desirable property of (2.7) is that it is still valid in the extreme case of zero translation since all vectors $\mathbf{f}_i \times \mathbf{R}\mathbf{f}'_i$ will become zero.

2.4.2 Coplanarity constraint in rolling shutter cameras

In rolling shutter cameras, however, the viewpoint is not unique for the features captured in the same frame. Here I propose a generalized coplanarity constraint for rolling shutter cameras.

First note that both the traditional epipolar constraint (2.6) and the coplanarity constraint (2.7) are expressed in terms of one of the two viewpoints. In fact, this frame of reference can be chosen arbitrarily. Once the reference is fixed, we can represent the camera orientation corresponding to any feature (determined by its exposure moment for rolling shutter cameras) in this reference. For the matched features between any two consecutive frames in rolling shutter cameras, I propose the following constraint

$$\det[(\mathbf{R}_1\mathbf{f}_1 \times \mathbf{R}'_1\mathbf{f}'_1)|(\mathbf{R}_2\mathbf{f}_2 \times \mathbf{R}'_2\mathbf{f}'_2)|(\mathbf{R}_3\mathbf{f}_3 \times \mathbf{R}'_3\mathbf{f}'_3)] = 0. \tag{2.8}$$

Note that in (2.8) $\mathbf{R}'_1$ means the camera orientation corresponding to feature 1 in the second frame, and not the transpose of $\mathbf{R}_1$. Constraint (2.8) does not

exactly hold in general cases but only under the assumption that the relative camera translations between the exposure moments for all pair of matched features are in the same direction. The readout time of two consecutive frames is at most 66ms (for 30 fps videos) and in such short period of time the camera translation can be well approximated by a constant direction. Note that such approximation is more general than the approximation used in [53] which assumes the linear velocity (both direction and magnitude) of the camera is constant. The constraint is illustrated by Fig. 2.3. In Fig. 2.3, the first three (from left to right) frames of axes correspond to the three features detected in the current frame. The last three frames of axes correspond to the three matched features in the next frame. The different orientations of the frames of axes show the changes in camera rotation while the features are exposed. The camera translation is approximated as the dashed ray. The three pairs of matched features are represented by green, blue, and orange arrows, respectively. By the proposed coplanarity constraint in rolling shutter cameras, the cross products of all pairs of matched features are perpendicular to the camera translation vector.

To make the constraint (2.8) more accurate I further apply such constraint only to groups of features that are not very far from each other in their y-axis coordinates. Based on (2.5) the exposure moments of features are close to each other if their y-axis coordinates are close. Assuming features $\mathbf{f}_1$, $\mathbf{f}'_1$, $\mathbf{f}_2$, $\mathbf{f}'_2$, $\mathbf{f}_3$, $\mathbf{f}'_3$ are selected in this way. Then the exposure moment difference among features in the same frame is much smaller compared to the exposure moment

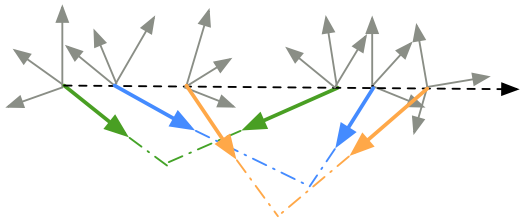


Figure 2.3: Coplanarity constraint in rolling shutter cameras. The cross products of all pairs of matched features are perpendicular to the camera translation vector.

difference between features in adjacent frames ($\approx$ 33ms). In this way, the camera translation vectors for the three pairs of features naturally have almost the same direction. Constraint (2.8) is less dependent on the constant-direction assumption in camera translation between two consecutive frames.

I use the coplanarity constraint (2.8) as implicit measurement to estimate all the parameters in an EKF. The way to represent the camera orientation corresponding to each feature using the parameters and gyroscope readings is shown in the next section.

2.5 EKF-based online calibration and synchronization

The online calibration and synchronization is based on an extended Kalman filter. The proposed EKF evolves when every video frame is captured, as in [38]. The state vector is defined as

$$\mathbf{x} = [f \; c_x \; c_y \; \kappa_1 \; \kappa_2 \; t_r \; t_d \; \mathbf{b}_g^{\mathrm{T}} \; \mathbf{q}_c^{\mathrm{T}}]^{\mathrm{T}}. \tag{2.9}$$

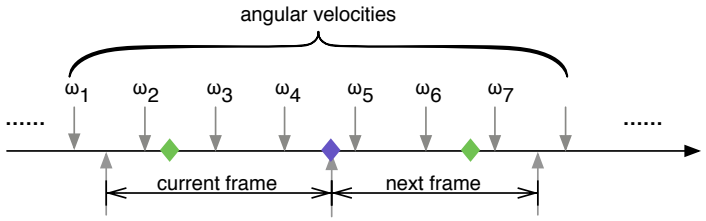


Figure 2.4: Timing relationship between the gyroscope readings and the video frames.

The gyroscope in cellphones usually has a higher sampling rate than the video frame rate. Moreover, timestamps of gyroscope readings and the video frames are not aligned. I show how to compute the relative rotation corresponding to each detected feature using gyroscope readings as following

2.5.1 Computation of relative rotation

Fig. 2.4 illustrates the timing relationship between the gyroscope readings and the video frames. Assume a pair of matched features $\mathbf{f}_i$ and $\mathbf{f}_i'$ are detected as at moments denoted by green diamonds and the reference time is fixed as the timestamp of the next frame (shown as the purple diamond). The relative camera orientation between the reference time and the exposure moment of a certain feature can then be expressed by the angular velocities

$$\mathbf{R}_i = \prod_{n=1}^{M} \Theta(\boldsymbol{\omega}_n \Delta t_n^i), \tag{2.10}$$

where M is the total number of angular velocities involved in computing the relative orientation (M=7 for the example shown in Fig. 2.4) and Δt_n^i is the time duration that the angular velocity $\boldsymbol{\omega}_n$ is used in the integration (assuming

constant angular velocity between readings). Note that not all of the M angular velocities have non-zero Δt_n^i values. For example, assume the timestamp of each angular velocity $\boldsymbol{\omega}_n$ is τ_n. Then for the feature in the next frame (right green diamond) in Fig. 2.4, only Δt_4^i, Δt_5^i and Δt_6^i are non-zero and they can be computed as

$$\begin{cases} \Delta t_4^i = \tau_5 - (T_{next} + t_d) \\ \Delta t_5^i = \tau_6 - \tau_5 \\ \Delta t_6^i = (T_{next} + t_d + t_r \frac{u_{y_i}}{h}) - \tau_6 \end{cases}, \tag{2.11}$$

where T_{next} is the framestamp of the next frame ($(T_{next} + t_d)$ corresponds to the moment for the purple diamond) and u_{y_i} is the y-axis coordinate of the feature ($(T_{next} + t_d + t_r \frac{u_{y_i}}{h})$ corresponds to the moment for the green diamond).

Each sub-relative rotation matrix can be computed by exponentiating the skew symmetric matrix formed by the product of angular velocity and its duration:

$$\Theta(\boldsymbol{\omega}_n \Delta t_n^i) = \exp(\text{skew}(\boldsymbol{\omega}_n) \Delta t_n^i), \tag{2.12}$$

where

$$\text{skew}(\boldsymbol{\omega}_n) = \begin{bmatrix} 0 & -\omega_{z_n} & \omega_{y_n} \\ \omega_{z_n} & 0 & -\omega_{x_n} \\ -\omega_{y_n} & \omega_{x_n} & 0 \end{bmatrix}. \tag{2.13}$$

Δt_n^i is determined by the exposure moments of $\mathbf{f}_i$ and $\mathbf{f}_i'$ computed using (2.5), and thus depends on the estimation variables t_r and t_d. The true angular velocities are represented as

$$\boldsymbol{\omega}_n = \hat{\boldsymbol{\omega}}_n + \mathbf{b}_g + \mathbf{n}_{g_n}, \tag{2.14}$$

where $\hat{\boldsymbol{\omega}}_n$ is the gyroscope reading, $\mathbf{b}_g$ is the gyroscope bias (to be estimated), and $\mathbf{n}_{g_n} \sim \mathcal{N}(0; \sigma_{\mathbf{g}})$ is the Gaussian distributed gyroscope measurement noise.

In this way, the relative camera orientation corresponding to any feature detected in the current and next frame can be expressed by the angular velocities.

2.5.2 State dynamics

All the parameters appeared in Section 2.3 except $\mathbf{b}_g$ are constant so they are just copied in state dynamics

$$\begin{cases} f(k+1) = f(k) \\ c_x(k+1) = c_x(k) \\ c_y(k+1) = c_y(k) \\ \kappa_1(k+1) = \kappa_1(k) \\ \kappa_2(k+1) = \kappa_2(k) \\ t_r(k+1) = t_r(k) \\ t_d(k+1) = t_d(k) \\ \mathbf{q}_c(k+1) = \mathbf{q}_c(k). \end{cases} \tag{2.15}$$

I model the dynamics of $\mathbf{b}_g$ by a random-walk process

$$\mathbf{b}_g(k+1) = \mathbf{b}_g(k) + \mathbf{m}_g(k), \tag{2.16}$$

where the random walk step $\mathbf{m}_g(k)$ is Gaussian distributed with zero mean and variance $\sigma_{\mathbf{b}}$.

2.5.3 State measurements

After features are matched between the current frame and the next frame, I picked N groups of features with three features in each group (without overlap). As mentioned in Section 2.4.2, to make the coplanarity constraint

more accurate, the selection of groups of features are not completely random. The three features in the same group should have close y-axis coordinates but relatively far away x-axis coordinates.

In this way we can obtain N measurements from the coplanarity constraint shown in Section 2.4. For instance, the measurement formed by features 1,2 and 3 is

$$0 = \det[(\mathbf{R}_1\mathbf{f}_1 \times \mathbf{R}'_1\mathbf{f}'_1)|(\mathbf{R}_2\mathbf{f}_2 \times \mathbf{R}'_2\mathbf{f}'_2)|(\mathbf{R}_3\mathbf{f}_3 \times \mathbf{R}'_3\mathbf{f}'_3)]. \tag{2.17}$$

The 3D feature locations $\mathbf{f}_i$ are computed by inverting the camera projection (2.3) as

$$\mathbf{f}_i = \mathbf{q}_c(k) \otimes \frac{1}{\varphi_i} \begin{bmatrix} (1 + \kappa_1 r^2 + \kappa_2 r^4)(u_{x_i} + v_{x_i} - c_x(k)) \\ (1 + \kappa_1 r^2 + \kappa_2 r^4)(u_{y_i} + v_{y_i} - c_y(k)) \\ f(k) \end{bmatrix}, \tag{2.18}$$

where $\mathbf{q}_c(k) \otimes (\cdot)$ means rotating a vector using 3D rotation defined by $\mathbf{q}_c(k)$, and φ_i is a normalization factor to make the result have unit norm. Besides normalization, there are two differences between (2.3) and (2.18): (a) I take the feature detection error $v_{x_i}, v_{y_i} \sim \mathcal{N}(0; \sigma_f)$ into account, and (b) the 3D feature is represented in the gyroscope coordinate system by multiplying the relative rotation estimate $\mathbf{q}_c(k)$ at stage k.

The relative rotation matrix $\mathbf{R}_i$ is computed according to (2.10). In this way, the right hand side of (2.17) can be expressed as a function of the state variables.

All of the N coplanarity constraints generates N implicit measurements

at a stage k according to (2.17)

$$\begin{cases} 0 = z_1(k) = h_1(\mathbf{x}(k), \{\hat{\boldsymbol{\omega}}_n\}, \{\mathbf{u}_i\}, \{\mathbf{n}_{gn}\}, \{\mathbf{v}_i\}), \\ 0 = z_2(k) = h_2(\mathbf{x}(k), \{\hat{\boldsymbol{\omega}}_n\}, \{\mathbf{u}_i\}, \{\mathbf{n}_{gn}\}, \{\mathbf{v}_i\}), \\ \qquad\qquad \vdots \\ 0 = z_N(k) = h_N(\mathbf{x}(k), \{\hat{\boldsymbol{\omega}}_n\}, \{\mathbf{u}_i\}, \{\mathbf{n}_{gn}\}, \{\mathbf{v}_i\}), \end{cases} \tag{2.19}$$

where $\{\hat{\boldsymbol{\omega}}_n\}$ are the gyroscope readings during the exposure time of two consecutive frames, $\{\mathbf{u}_i\}$ are the 2D coordinates of all of the observed features. $\{\mathbf{n}_{gn}\}$ and $\{\mathbf{v}_i\}$ are the gyroscope measurement noise and feature observation noise, respectively, as shown in (2.14) and (2.18). Please note in the measurement equations the measurement noise appears implicitly as non-additive noise.

The state update is performed right after state prediction. Only one round of state prediction and update is needed once a new frame is read and all features are tracked.

2.5.4 Extended Kalman filter computation

In EKF state vector estimate is predicted using dynamic equations and then updated using measurement equations. Prediction and Update rely on the Jacobian matrices of the dynamic and measurement equations with respect to the state vector and the system noise. The linear dynamic equations (2.15) and (2.16) lead to very simple Jacobian matrices (identity matrix). The Jacobian matrices of the measurement equations can also be computed analytically in closed-form. I show the derivations in Appendix 1.

The camera-to-gyroscope orientation is represented by unit quaternion $\mathbf{q}_c$. Traditional extended Kalman filter cannot guarantee unit norm of the quaternion after estimate update. Therefore we use a minimal 3-element representation $\delta\boldsymbol{\theta}$ for the estimate error of $\mathbf{q}_c$ as in [87]. The true value of $\mathbf{q}_c$ can be represented as

$$\mathbf{q}_c = \delta\mathbf{q} \otimes \hat{\mathbf{q}}_c, \tag{2.20}$$

where $\hat{\mathbf{q}}_c$ is the estimate and

$$\delta\mathbf{q} = \begin{bmatrix} \delta\boldsymbol{\theta}/2 \\ \sqrt{1 - ||\delta\boldsymbol{\theta}/2||_2^2} \end{bmatrix}. \tag{2.21}$$

With such error representation we can update the estimate in a multiplicative way and guarantee the unit norm of the estimate. For more details please see [87].

In practice EKF update is executed every other frame (or less often to reduce complexity). The reason is that the measurement equation (2.17) involve features detected from two consecutive frames. If EKF is updated every frame then the features in each frame are used twice, which causes correlation between feature detection errors and the state estimate. One can augment the state vector to track the feature detection errors. However, such augmentation will further increase the computational burden, while updating state estimate every other frame can easily avoid such correlation without augmenting the state vector.

2.5.5 State initialization

The state vector needs to be initialized carefully to make the EKF work properly. I initialize the principal point coordinates c_x, c_y to be the center of the frame. The focal length is initialized using the horizontal view angle provided by the smart phone operating system. If the operation system of the smartphone does not provide the value of horizontal view angle, SOG filters can be used with several initial guesses as in [10]. The readout time t_r is initialized as 0.0275 ms which is about 82.5% of the entire interval between frames. The coordinate system of the gyroscope is defined relative to the screen of the phone in its default orientation in all Android phones. Thus we can obtain the initial guess of $\mathbf{q}_c$ depending on whether front or rear camera is being used. This initial guess is usually accurate enough, but the proposed calibration is necessary since the camera is sometimes not perfectly aligned with the screen of the phone. The initial values of all other parameters (t_d and $\mathbf{b}_g$) are just set as 0.

To make sure that the true value lies in the 3σ intervals of the initial Gaussian distributions, I initialize the standard deviation of c_x, c_y, f, t_r as 6.67 pixels, 6.67 pixels, 20 pixels, and 0.00167 s, respectively. t_d is initialized as a sum of Gaussian distribution because of the highly non-linearity of the measurements with respect to t_d. The set of Gaussian distributions are initialized uniformly in the range of $\pm 30ms$. The standard deviation of each element in $\mathbf{b}_g$ is initialized as 0.006. The standard deviation of the estimate error of $\mathbf{q}_c$ is initialized as 0.5 degrees along each axis. The standard deviation of the

radial distortion parameters κ_1 and κ_2 is initialized as 0.1. I set the standard deviation of gyroscope measurement noise and feature detection error as 0.003 rads/s and 1 pixels, respectively. The standard deviation of gyroscope measurement noise is determined from computing the reading variance while the cellphone is put still. Due to the sum-of-Gaussian initialization of t_d, the algorithm starts from a SOG filter but it quickly converges to a single EKF using pruning of distributions with low weights [10].

2.6 Experimental results

In this section I test the proposed algorithm with both Monte Carlo simulation and cellphone experiments.

2.6.1 Monte Carlo simulation

In the Monte Carlo simulation we randomly locate 1000 3D feature points distributed in range $X \in [-30, 30]$ meters, $Y \in [-20, 20]$ meters, $Z \in [30, 60]$ meters, respectively. The ground truth value of the parameters are set as $f = 690$ pixels, $c_x = 355$ pixels, $c_y = 220$ pixels, $\kappa_1 = 0.111$, $\kappa_2 = -0.303$, $t_r = 0.02$ s, $t_d = 0.02$ s, $\mathbf{q}_c = [\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, 0]^{\mathrm{T}}$ respectively. $\mathbf{b}_g$ is initialized as $[-0.008, 0.002, 0.017]^{\mathrm{T}}$ rads/s and then simulated by random walk. All of these values come from the parameters of a real cellphone camera. The ground truth motion of camera is fixed with a randomly generated sequence of angular velocities and linear velocities. The angular velocity and linear velocity sampling rate is set as 100 Hz. With the ground truth motion and

camera/gyroscope parameters, I artificially generate a video with 250 frames at frame rate 30fps. Note that each video frame is not a real image but a sparse 2D point cloud.

In each trial of Monte Carlo simulation I generate Gaussian random gyroscope measurement noise and feature detection errors according to the variances shown in Section 2.5.5. In this way, I can artificially add the noise and simulate the gyroscope readings and feature detections. Then I run EKF calibration in each trial, with state estimate initialized randomly within 3σ range around the ground truth values (note that this initialization method is different from that in Section 2.5.5, which is used for cellphone experiments). In state update I use only 150 virtual features (50 measurements) picked from the feature pool.

I run 50 Monte Carlo trials to compare the proposed online estimation with the online estimation proposed in my earlier work [40]. The proposed estimation differs from [40] primarily in lens distortion modeling, Jacobian matrices computation ([40] computed them numerically) and selection of features ([40] selected the features completely randomly without considering the y-axis coordinate distance). I also compare the online calibration with a batch optimization using all of the frames. The batch optimization is solved via Levenberg-Marquardt algorithm [64].

Table 2.1 shows the root mean square (RMS) error of the parameter estimation before calibration and after calibration (with 250 frames). The estimation error of the gyroscope bias $\mathbf{b}_g$ is not shown since it is time-varying.

Table 2.1: RMS error of 50 Monte Carlo simulation trials.

RMS error				
state variable	before calibration	batch optimization	online estimation	[40]
f (pixels)	17.3	0.910	2.28	3.70
c_x (pixels)	8.26	0.645	2.55	3.45
c_y (pixels)	6.55	0.576	0.96	2.40
t_r (ms)	1.8	0.031	0.078	0.15
t_d (ms)	17.1	0.027	0.041	0.089
$\mathbf{q}_c$ (degrees)	0.60	0.076	0.196	0.285
κ_1	0.099	0.0014	0.0042	N/A
κ_2	0.076	0.0026	0.0060	N/A

The estimation error of $\mathbf{q}_c$ is converted to a single angle (computed as the L^2 norm of the minimal 3-element error representation). We can find that batch optimization performs the best. The proposed EKF-based calibration method is also able to successfully converge to the ground truth value. With the modifications proposed in this dissertation, we can achieve a better calibration compared with [40]. Please note that although batch optimization gives the closest estimate, the EKF-based online calibration can be implemented in real time and enable immediate use of gyroscope in vision applications. More importantly, online calibration is able to deal with time-varying parameters, such as varying f due to zoom and varying t_d due to clock drift.

In Fig. 2.5 I show the estimation error along EKF-based calibration in one trial, with blue lines representing the estimation error and red lines representing the 99.7% (3σ) uncertainty bounds. For the relative orientation

$\mathbf{q}_c$ we only show the estimation error after converting to a single angle as in table 2.1. From Fig. 2.5 we can observe that the proposed method is able to accurately estimate the parameters.

2.6.2 Cellphone experiments

In the cellphone experiments, I use a Google Nexus S Android smartphone that is equipped with a three-axis gyroscope. We capture the videos and the gyroscope readings from the cellphone and run the proposed online calibration and synchronization in MATLAB. The feature points are tracked using KLT tracker. I divide the frame into 4 equally sized bins and perform outlier rejection locally within each bin by computing a homography transformation using RANSAC [21], as in [28]. I estimate the ground truth of camera projection parameters (with lens distortion) using the offline camera calibration method in [93]. The ground truth of timestamp delay t_d is obtained by offline calibration in [36]. The ground truth of rolling shutter readout time t_r is obtained by batch optimization under pure rotational camera motion as in [46]. The estimated values are not guaranteed to be equal to the ground truth values so I only use them as a reference to roughly examine the accuracy of the proposed algorithm. I test the performance of the proposed method on various video sequences and show the results on two typical sequences: one shot while running forward and the other shot while panning the camera in front of a building. Fig. 2.6 shows two frames extracted from the two test

Table 2.2: Absolute estimation error for the running sequence.

Absolute estimation error		
state variable	before calibration	after online calibration
f (pixels)	26.5	4.16
c_x (pixels)	5.2	2.32
c_y (pixels)	13.57	1.50
t_r (ms)	3.72	0.21
t_d (ms)	13.2	0.14
κ_1	0.111	0.012
κ_2	0.303	0.045

sequences [1].

The running sequence (with 250 frames) is used to test the performance of the algorithm under arbitrary camera motion, including very high frequency shake and non-zero translation. The absolute estimation errors before and after online calibration and synchronization are shown in Table 2.2. We can observe that the proposed method is able to estimate the parameters that are close to offline separate calibration.

In the second test video sequence (with 241 frames) I simply pan the camera in front of a building. This video is used to test the algorithm under (almost) zero camera translation (pure rotation). The estimation errors are shown in Table. 2.3. The proposed algorithm works equally well compared to the running sequence.

[1]The videos can be found at `http://users.ece.utexas.edu/~bevans/papers/2015/autocalibration/`.

Table 2.3: Absolute estimation error for the panning sequence.

Absolute estimation error		
state variable	before calibration	after online calibration
f (pixels)	26.5	3.57
c_x (pixels)	5.2	1.04
c_y (pixels)	13.57	2.29
t_r (ms)	3.72	0.056
t_d (ms)	21.7	0.33
κ_1	0.111	0.014
κ_2	0.303	0.055

To better display the difference before and after synchronization of the timestamps between video frames and gyroscope readings, I show the rates of 2D translation of pixels compared to the gyroscope data as in [46]. If we ignore the rolling shutter effect and the camera rotation around z-axis, the average rate of pixel translation can be approximated as

$$\begin{cases} \dot{u_x}(t) \approx f \cdot \omega_y(t + t_d) \\ \dot{u_y}(t) - \approx f \cdot \omega_x(t + t_d), \end{cases} \tag{2.22}$$

where $\omega_x(t)$ and $\omega_y(t)$ are angular velocities around x-axis and y-axis. These two angular velocity sequences can be obtained discretely from the gyroscope readings (after adding the gyroscope bias and transformed by $\mathbf{q}_c$). The pixel translation rate on the left hand side of (2.22) is approximated by finite differences between consecutive frames. In Fig. 2.7 and Fig. 2.8 I show the pixel translation rates and the angular velocities (right hand side of (2.22)) for the running sequence. I only plot a 3-second duration sequence in order to make

the difference look more obvious. We can observe that after calibration and synchronization, the curve from video data and gyro data align much better, which indicates the effectiveness of the proposed algorithm.

In Fig. 2.9 and Fig. 2.10 I show the same comparison for the panning sequence. Again, the pixel translations computed from the video and gyroscope readings align very well after the proposed online calibration and synchronization.

2.6.3 Rolling shutter artifact rectification after calibration

I apply the proposed online calibration and synchronization algorithm in rectifying the rolling shutter artifact in video sequences. After calibration and synchronization the camera rotation can be directly obtained from gyroscope readings. The rolling shutter artifact is rectified by warping each row in the frame so that all of the rows are captured at the same moment (we fix this moment as the starting time of each frame). Fig. 2.11 and Fig. 2.12 show that the gyroscope readings can effectively correct the rolling shutter artifact after sensor calibration.

2.6.4 Run time

The current running speed of the proposed algorithm implemented in MATLAB (where feature detection and tracking are implemented using mex functions of an OpenCV implementation [91]) is 20.95 fps on a laptop with 2.3GHz Intel i5 processor. In the simulation, I had run the algorithm on every

other pair of adjacent frames. However, we can run the calibration less often than using every other pair of adjacent frames, which allows a scaling back of the calculations to meet real-time constraints.

2.7 Conclusions

In this chapter I propose an online calibration and synchronization algorithm for cellphones that is able to estimate not only the camera projection parameters, but also the gyroscope bias, the relative orientation between the camera and gyroscope, and the delay between the timestamps of the two sensors. The proposed algorithm is based on the generalization of the coplanarity constraint of the cross products of matched features in a rolling shutter camera model. The proposed algorithm can also be naturally extended to a global shutter camera model by forcing the readout time for each frame t_r to be zero. Monte Carlo simulation and experiments run on real data collected from cellphones show that the proposed algorithm can successfully estimate all of the needed parameters with different kinds of motion of the cellphones. This online calibration and synchronization of rolling shutter camera and gyroscope make it more convenient for high quality video recording, gyro-aided feature tracking, and visual-inertial navigation.

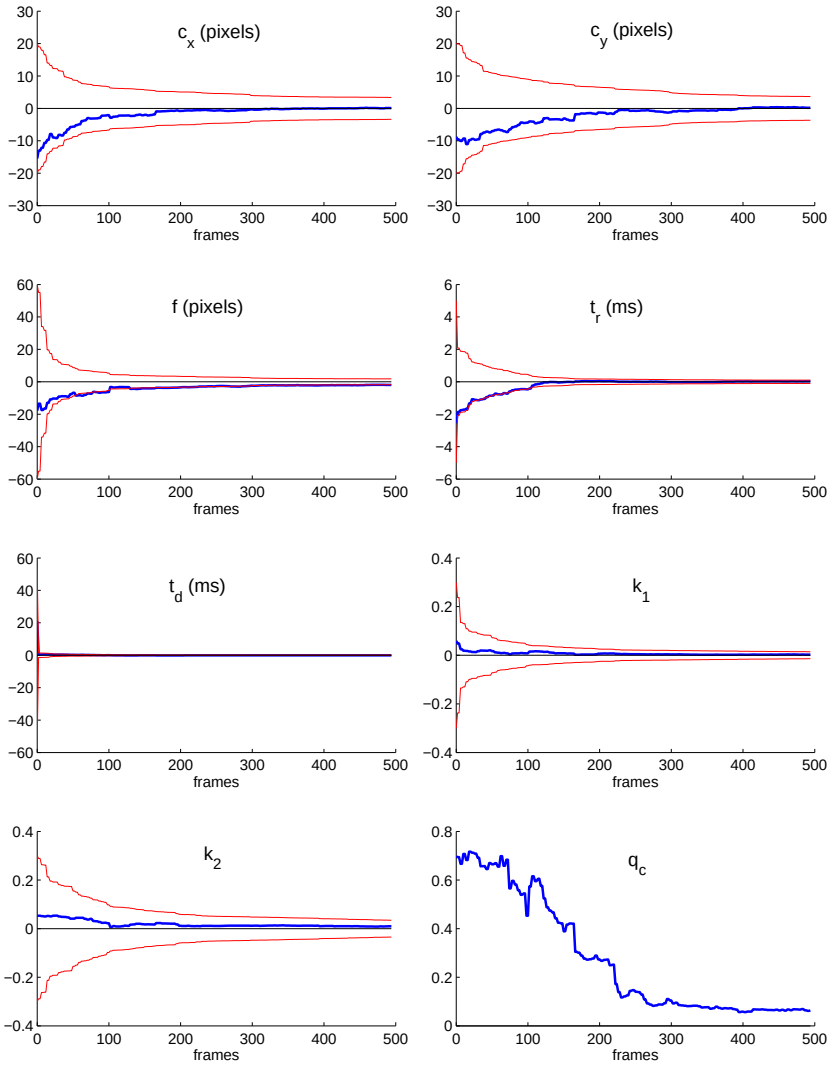
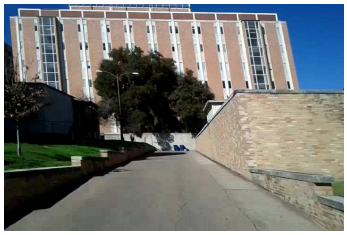


Figure 2.5: Estimation error over time in one Monte Carlo simulation trial.

Running sequence



Panning sequence

Figure 2.6: Examples of frames extracted from the test sequences.

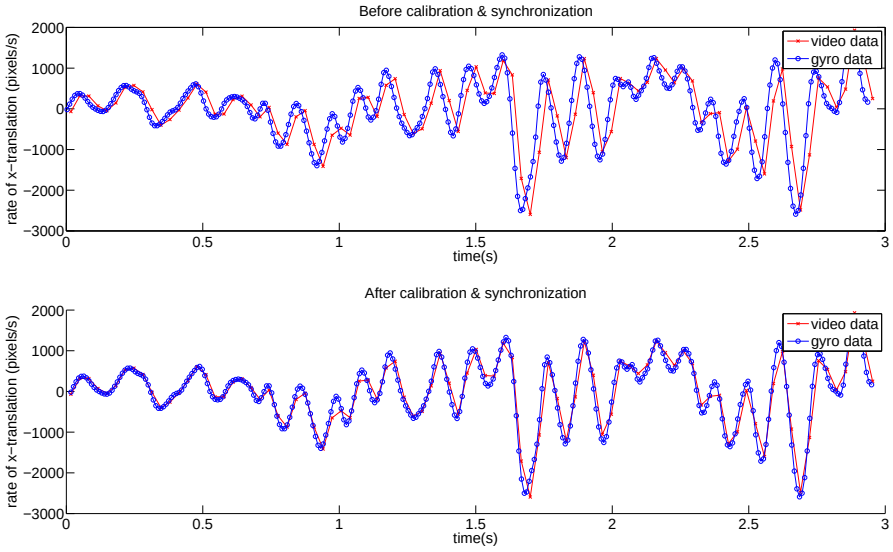


Figure 2.7: Horizontal pixel translation rate $\dot{u_x}(t)$ (red) and $f \cdot \omega_y(t+t_d)$ (blue) for the running sequence.

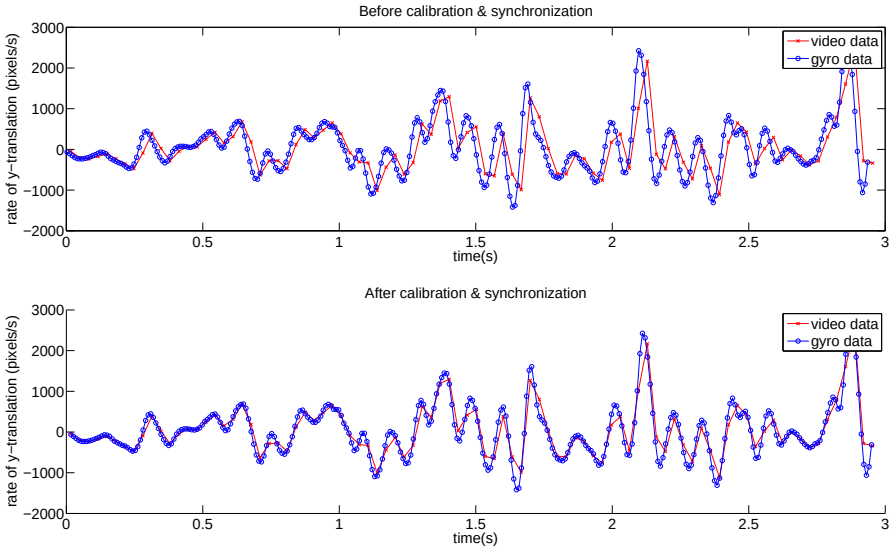


Figure 2.8: Vertical pixel translation rate $\dot{u}_y(t)$ (red) and $-f \cdot \omega_x(t+t_d)$ (blue) for the running sequence.

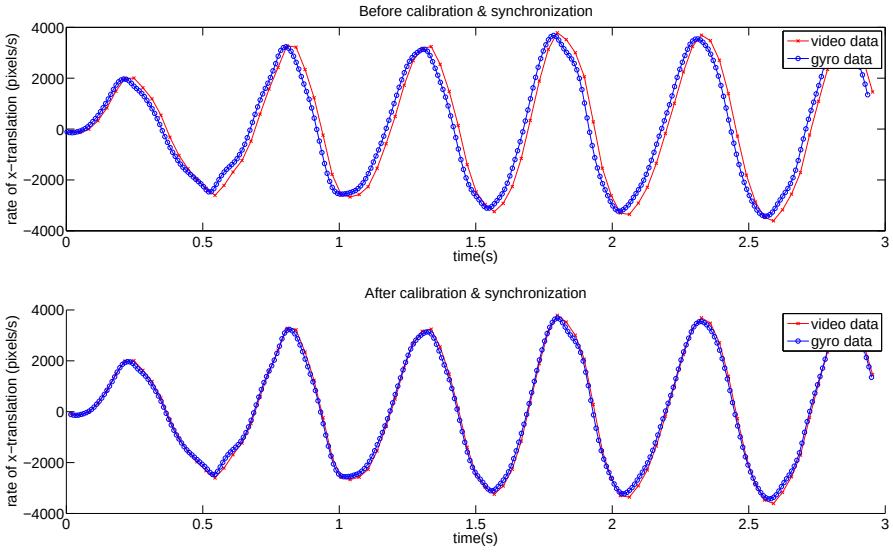


Figure 2.9: Horizontal pixel translation rate $\dot{u_x}(t)$ (red) and $f \cdot \omega_y(t+t_d)$ (blue) for the panning sequence.

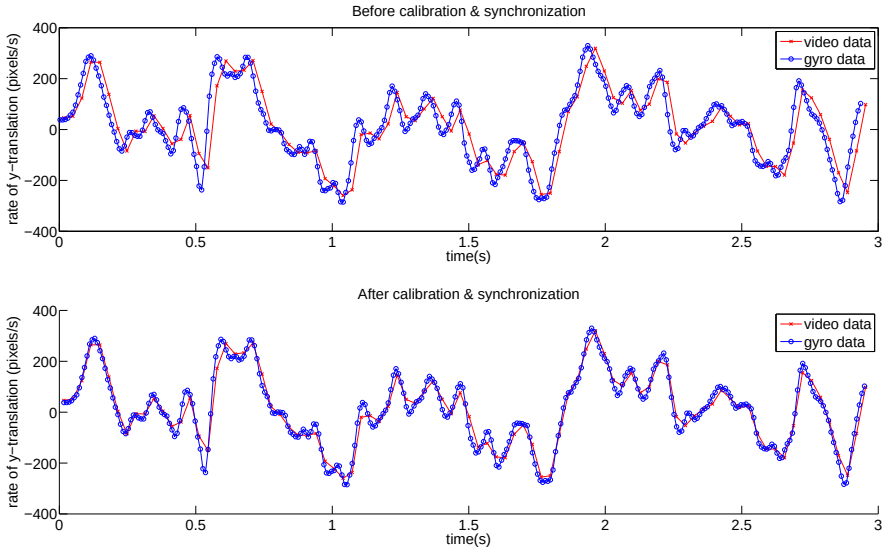


Figure 2.10: Vertical pixel translation rate $\dot{u_y}(t)$ (red) and $-f \cdot \omega_x(t+t_d)$ (blue) for the panning sequence.

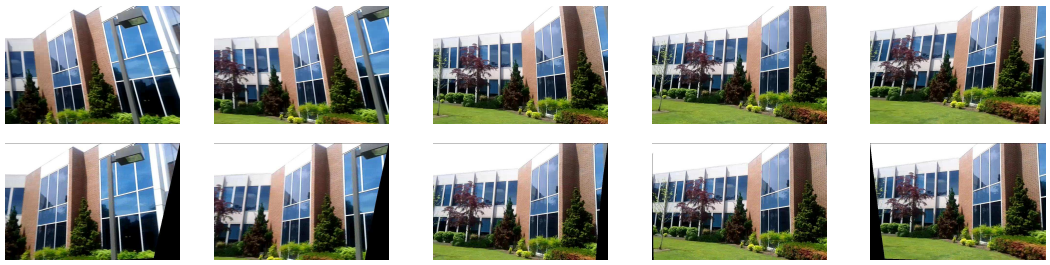


Figure 2.11: Rolling shutter artifact rectification for the running sequence using the gyroscope readings after sensor calibration and synchronization. Top: five consecutive frames with rolling shutter artifact. Bottom: the rectified frames.

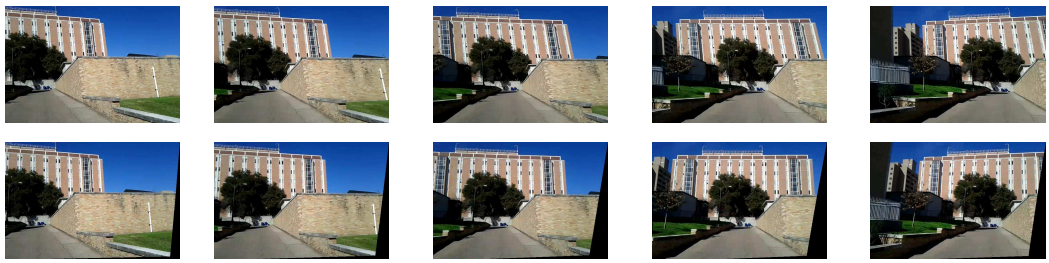


Figure 2.12: Rolling shutter artifact rectification for the panning sequence using the gyroscope readings after sensor calibration and synchronization. Top: five consecutive frames with rolling shutter artifact. Bottom: the rectified frames.

Chapter 3

Constrained 3D Rotation Smoothing via Global Manifold Regression

3.1 Introduction

In chapter I focus on the second step of video stabilization: motion smoothing. Specifically, I solve the problem of offline motion smoothing.

The proposed motion smoothing algorithm is based on a 3D rotational camera motion model for a calibrated camera with a known intrinsic matrix. Compared to 2D affine or projective motion models, 3D motion models can more accurately reflect the real camera perspective projection, and thus give more realistic motion smoothing and avoid image distortion in frame synthesis. I only smooth 3D rotation of the camera instead of both rotation and translation because (1) the unwanted jitter in videos are primarily caused by camera rotation, and (2) frame synthesis with 3D camera translation would need the depth value at every pixel, which is very difficult to obtain accurately. To estimate the 3D camera rotation I use a gyroscope after sensor calibration using the algorithm in Chapter 2. Current gyroscopes in smart phones have very high precision and can return more reliable 3D camera rotation estimates compared to the estimates obtained from visual features in the video sequence,

especially when there are many moving objects in the scene or it is difficult to track feature points due to motion blur or illumination changes.

Under a 3D rotational model, camera motion for a video can be considered as a sequence of 3D rotation matrices. I formulate motion smoothing as a regression problem with a regularization term indicating the smoothness of the sequence of rotation matrices. Unlike traditional approaches, I exploit the manifold structure of the sequence of rotation matrices. The formulated problem is based on geodesic distance on the Riemannian manifold.

Due to the change of camera poses introduced by video stabilization, the stabilized frames can be only synthesized for portions of the scene that are visible in the original frames. Therefore, we have to crop the resulting video with a large enough cropping size to keep most of the content of the original video sequence while at the same time guaranteeing that no black borders intrude into the stabilized video frames. In this chapter, I introduce a geodesic-convex constraint on the manifold to approximate such requirement so that the entire motion smoothing problem is kept geodesic-convex on the manifold.

Previous methods have only exploited the properties on the manifold of the individual 3D rotation matrix $\mathbf{SO(3)}$ (Special Orthogonal Group), so they can only smooth the camera motion locally through low-pass filtering. Considering the entire set of sequences of rotation matrices as a Riemannian manifold allows me to model the motion smoothing problem globally with proper constraints and solve it optimally.

To solve the formulated constrained smoothing problem on the manifold, I compute the gradient and Hessian of the objective function using Riemannian geometry, and then extend the two-metric projection algorithm in Euclidean space to non-linear manifolds. The proposed manifold optimization algorithm has much better convergence property than normal non-linear optimization algorithms in Euclidean space. Experimental results show that my motion smoothing method outperforms state-of-the-art methods by generating more stable videos with less distortion.

This chapter is organized as follows: Section 3.2 reviews previous video stabilization algorithms and related optimization background. Section 3.3 formulates motion smoothing as a regression problem on the sequence of rotation matrices using geodesic distance. Section 3.4 adds hard geodesic-convex constraints to the optimization problem to guarantee that no black border will be present in the stabilized videos. Section 3.5 presents the computation of gradient and Hessian of the objective function using Riemannian geometry and then generalizes two-metric projection algorithm in Euclidean space to non-linear manifolds. Section 3.6 shows the convergence of the proposed algorithms and compares the proposed motion smoothing method against state-of-the-art algorithms. Section 3.7 concludes the chapter. I have publicly released the Matlab code for video stabilization using the proposed motion smoothing algorithm [37].

3.2 Related Work

Camera motion has been commonly modeled using 2D affine or projective approaches [29, 34, 62]. Using full 3D models including both rotation and translation for calibrated cameras was first proposed in [8] and further discussed in [56]. In both papers complicated approximations are used in frame synthesis to handle the problem of missing depth values. In [46, 65] pure 3D rotational models with known intrinsic camera parameters were shown to generate high-quality results while only needing homography-based warping in frame synthesis.

Gyroscopes and other inertial measurement sensors have been widely used in robotic localization problems together with visual measurements [43, 83]. However, they were not used in video stabilization to replace the feature-based motion estimation until they became accurate enough and widely available in cell phones recently [31, 46]. Compared with camera motion estimation using only visual measurements [12, 74], estimation with inertial measurements is faster and more robust, especially for the cell phone cameras that use CMOS image sensors (with rolling shutters). Using the camera motion estimated with the help of the inertial measurements [31, 38, 46], rolling shutter effect can be effectively rectified so that each frame looks as captured under a single camera pose. For the rest of the chapter, I assume that any possible rolling shutter effects are rectified before video stabilization is applied. My proposed video stabilization methods would therefore work for cameras with or without rolling shutters.

Motion smoothing methods using 2D models are based on Euclidean distance. 2D camera motion can be smoothed using local methods such as Gaussian-kernel low-pass filtering [62], global methods such as L^1-based regularization [29], and real-time methods such as Kalman filtering [55]. 3D rotation smoothing has been implemented locally by low-pass filtering based on either Euclidean distance [46] or geodesic distance on the manifold $\mathbf{SO(3)}$ [31, 56]. Motion smoothing has also been performed directly on the feature trajectories without explicitly estimating the parametric camera motion [51, 57, 90]. These methods are actually also based on 2D motion models.

The manifold structure of 3D rotation has been extensively studied in computer graphics. It has been shown that a linear interpolation on the geodesic between two different poses gives a very smooth and natural animation of rigid body [77]. Such interpolation is equivalent to constructing a curve that minimizes the sum of geodesic distances between every pair of adjacent knots. This fact motivates my formulation of camera motion smoothing on the manifold and the use of geodesic distance as the smoothness metric.

Although $\mathbf{SO(3)}$ has additional applications in computer vision [22, 27], the sequence of 3D rotation matrices was hardly investigated as a whole. In [7] discrete regression is first applied on the sequence of rotation matrices with conjugate gradient descent algorithm proposed to solve the formulated problem. In this chapter I also directly exploit the manifold structure of sequences of rotation matrices so that we can formulate 3D rotation smoothing as a regression problem. Compared to [7], I further compute the Hessian of the

objective function using Riemannian geometry so that the problem can be solved more efficiently using Newton's method on the manifold.

Previous video stabilization methods usually stabilize the video first without considering the cropping size of the result and crop the stabilized video as a post-processing step [31, 56]. Such methods cannot optimally smooth a video sequence with a pre-fixed cropping size and usually have to sacrifice the smoothness. Rendering the unseen part of the frame using mosaicking and inpainting algorithms with the help of neighboring frames allows the original size of the video to be kept [62]. However, the rendered parts usually have much lower image quality, especially for the videos with a lot of moving objects. The cropping size is first considered as a hard constraint in motion smoothing step in [29]. In this chapter I approximate this constraint with a geodesic-convex set on the manifold. Constrained optimization on Euclidean space has been extensively studied [6], but not on non-linear manifolds. If the constraint set has some simple structure, such as a Cartesian product of Euclidean balls, an efficient two-metric projection algorithm can be used to solve the optimization problem [13, 24]. The proposed constraint set in this chapter is a Cartesian product of geodesic balls on manifold. I extend the two-metric projection algorithm in Euclidean space to general manifolds so the proposed manifold optimization problem can be solved efficiently and optimally. Table 3.1 summarizes motion smoothing methods in prior work and in this chapter for video stabilization.

Table 3.1: Comparison of prior work and this chapter on offline motion smoothing for video stabilization.

Paper	Motion model	Smoothing method	Constrained by cropping size	Global smoothing
[62]	2D	Low-pass filtering	no (full-frame with inpainting)	no
[46]	3D (Euclidean)	Low-pass filtering	no	no
[56] [31]	3D (manifold)	Low-pass filtering	no	no
[29]	2D	Regression	yes	yes
[51]	2D (trajectories)	Regression	no	no
[57]	2D (trajectories)	Subspace low-pass filtering	no	no
Proposed	3D (manifold)	Regression	yes	yes

3.3 Smoothness of 3D Rotation Sequence

All of the 3×3 rotation matrices constitute the Special Orthogonal Group $\mathbf{SO(3)}$, in which any element $\mathbf{R}$ satisfies the constraints $\mathbf{R}\mathbf{R}^{\mathrm{T}} = \mathbf{I}$ and $\det(\mathbf{R}) = 1$. $\mathbf{SO(3)}$ can be also considered as an embedded Riemannian submanifold of Euclidean space $\mathbb{R}^9$ (represented as 3×3 real matrices). A natural extension of Euclidean distance in Euclidean space to the Riemannian manifold $\mathbf{SO(3)}$ is the geodesic distance

$$d_g(\mathbf{R}_i, \mathbf{R}_j) = ||\mathrm{logm}(\mathbf{R}_i^{\mathrm{T}}\mathbf{R}_j)||_F, \tag{3.1}$$

where $\mathrm{logm}(\cdot)$ is the matrix logarithm operator and $||\cdot||_F$ is the Frobenius norm of a matrix. In fact, $\mathrm{logm}(\mathbf{R}_i^{\mathrm{T}}\mathbf{R}_j)$ is a skew-symmetric matrix repre-

senting a tangent vector in the tangent space $T_{\mathbf{R}_i}\mathbf{SO}(\mathbf{3})$ that indicates the non-normalized direction from $\mathbf{R}_i$ to $\mathbf{R}_j$ on $\mathbf{SO}(\mathbf{3})$. Usually we also write $\text{logm}(\mathbf{R}_i'\mathbf{R}_j)$ as $\log_{\mathbf{R}_i}\mathbf{R}_j$ and call it the logarithmic mapping. Inversely, given any tangent vector $\xi \in T_{\mathbf{R}_i}\mathbf{SO}(\mathbf{3})$, we can define $\exp_{\mathbf{R}_i}\xi = \mathbf{R}_i\text{expm}(\xi)$, where $\text{expm}(\cdot)$ is the matrix exponential operator. Here, $\exp_{\mathbf{R}_i}\xi$ is called the exponential mapping and is used to move $\mathbf{R}_i$ along the direction defined by ξ on $\mathbf{SO}(\mathbf{3})$. The logarithmic mapping and exponential mapping together define a curve

$$t \in [0,1] \mapsto \gamma(t) = \exp_{\mathbf{R}_i}\left(t \cdot \log_{\mathbf{R}_i}\mathbf{R}_j\right), \tag{3.2}$$

which is known as the minimizing geodesic from $\mathbf{R}_i$ to $\mathbf{R}_j$ on $\mathbf{SO}(\mathbf{3})$. The minimizing geodesic is a generalization of the notion of "straight line" in Euclidean space to Riemannian manifolds, representing the shortest path between two points in the manifolds given a Riemannian metric. The length of the minimizing geodesic is defined in (4.1).

For each video sequence, we can obtain a sequence of 3D rotation matrices corresponding to all of the frames from the gyroscope readings or the estimation using matched feature points. Next I consider the sequence of 3D rotation matrices as a whole and exploit the properties of the Riemannian manifold constituted by these sequences.

Assume $\mathbf{x} = [\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_N]^{\mathrm{T}}$ represents the sequence of 3D camera rotation for any video sequence with N frames. Clearly all of the possible rotation matrix sequences with N elements constitute a manifold $\mathcal{M}_R$ with

dimension $3N$. In fact, we have

$$\mathcal{M}_R = \mathbf{SO(3)} \times \mathbf{SO(3)} \times \ldots \times \mathbf{SO(3)}, \tag{3.3}$$

a Cartesian product of N $\mathbf{SO(3)}$ manifolds. Furthermore, for any $\mathbf{x} \in \mathcal{M}_R$, the tangent space $T_{\mathbf{x}}\mathcal{M}_R$ at $\mathbf{x}$ can be represented as

$$[\eta_1, \eta_2, \ldots, \eta_N]^{\mathrm{T}}, \tag{3.4}$$

where $\{\eta_i\}$ are real skew-symmetric matrices. In other words, the tangent vectors and corresponding exponential (and logarithmic) mapping are still separable as the elements in the manifold of rotation matrix sequences. This makes the proposed gradient-related optimization algorithms in the next section easy to implement.

The goal of video stabilization is to remove the visible jitter and make the camera motion trajectory change smoothly. Given the manifold structure of $\mathbf{SO(3)}$, it is natural to define the smoothness of a rotation matrix sequence as the sum of geodesic distances between adjacent rotation matrices. At the same time, we need to guarantee that the smoothed camera motion trajectory does not deviate from the original trajectory too much. As a result, I formulate the video stabilization problem as

$$\min_{\{\mathbf{R}_i\}} \sum_{i=1}^{N} \frac{1}{2} d_g^2(\tilde{\mathbf{R}}_i, \mathbf{R}_i) + \alpha \sum_{i=1}^{N-1} \frac{1}{2} d_g^2(\mathbf{R}_i, \mathbf{R}_{i+1}), \tag{3.5}$$

where $\{\mathbf{R}_i\}$ is the sequence of stabilized rotation matrices, $\{\tilde{\mathbf{R}}_i\}$ is the original sequence of rotation matrices, α is the weighting parameter controlling the

smoothness of the stabilized trajectory. (3.5) is an extension of discrete curve fitting problem in Euclidean space with penalty on the first order difference. Note that although the objective function is derived based on the geodesic distance between elements in $\mathbf{SO(3)}$, it is defined on the rotation matrix sequence manifold $\mathcal{M}_R$.

3.4 Constrained video Stabilization

The proposed objective function in (3.5) is effective in smoothing the sequence of 3D rotation matrices. However, in the last step of video stabilization, the synthesized frames may contain black borders since not every pixel in the synthesized frame is visible in the original frame due to the change of camera orientation. Therefore, we have to crop the synthesized frames into a smaller size so that there are no black borders in the stabilized video. In other words, given a preferred stabilized size of the video, the video stabilization system must guarantee that every pixel in the cropped stabilized frames is visible in the original frames. This is a hard constraint that has to be considered in the camera motion smoothing algorithm.

Assume the intrinsic projection matrix of the camera is given as $\mathbf{K}$. Under pure rotational camera model, for any pixel $[u_{ij}, v_{ij}]^{\mathrm{T}}$ in the stabilized frame i, its corresponding 2D pixel location in the original frame $[\tilde{u}_{ij}, \tilde{v}_{ij}]^{\mathrm{T}}$ can be computed as

$$\begin{bmatrix} \tilde{u}_{ij} \\ \tilde{v}_{ij} \end{bmatrix} = g\left(\mathbf{K}\tilde{\mathbf{R}}_i {\mathbf{R}_i}^{\mathrm{T}} \mathbf{K}^{-1} \begin{bmatrix} u_{ij} \\ v_{ij} \\ 1 \end{bmatrix} \right), \tag{3.6}$$

where the function

$$g\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} x/z \\ y/z \end{bmatrix} \tag{3.7}$$

is used to convert the homogeneous coordinates into inhomogeneous coordinates. Assume that the frame size in the original video is $w \times h$, and the coordinates of the top left corner and bottom right corner of the cropped rectangle in the stabilized video are $[c_1, d_1], [c_2, d_2]$, the hard constraint for video stabilization can be represented as

$$\begin{cases} 0 \leq \tilde{u}_{ij} \leq w \\ 0 \leq \tilde{v}_{ij} \leq h \end{cases}, \forall \begin{bmatrix} u_{ij} \\ v_{ij} \end{bmatrix} s.t. \begin{cases} c_1 \leq u_{ij} \leq c_2 \\ d_1 \leq v_{ij} \leq d_2 \end{cases} \tag{3.8}$$

The constraint (4.5) is very complex with respect to the rotation matrices that we want to compute and no algorithms as far as I know are guaranteed to handle it efficiently (note that for 2D affine or similarity motion models, this constraint is just linear with respect to the variables). To overcome this difficulty I replace the constraint in (4.5) with a simpler constraint defined on the manifold

$$|| \log_{\tilde{\mathbf{R}}_i} \mathbf{R}_i ||_F \leq r_0, \forall i, \tag{3.9}$$

where r_0 is a fixed threshold depending on the relative size of the cropped rectangle in the stabilized frames. The constraint (3.9) just means that the geodesic distance between the original and stabilized camera orientations should be less than r_0. r_0 is defined as the largest value to guarantee that for all of the camera orientations satisfying the constraint (3.9), the constraint (4.5) is also satisfied. We know that any 3D rotation matrix can be represented by a

rotation axis and a rotation angle, so constraint (3.9) can be also interpreted as the rotation angle of $(\tilde{\mathbf{R}}_i)^{\mathrm{T}}\mathbf{R}_i^{new}$ being no larger than r_0. Constraint (3.9) is homogeneous on every possible rotation axis and is clearly stricter than constraint (4.5). Fig. 3.1 shows that the constraint (3.9) is a good approximation of the original constraint.

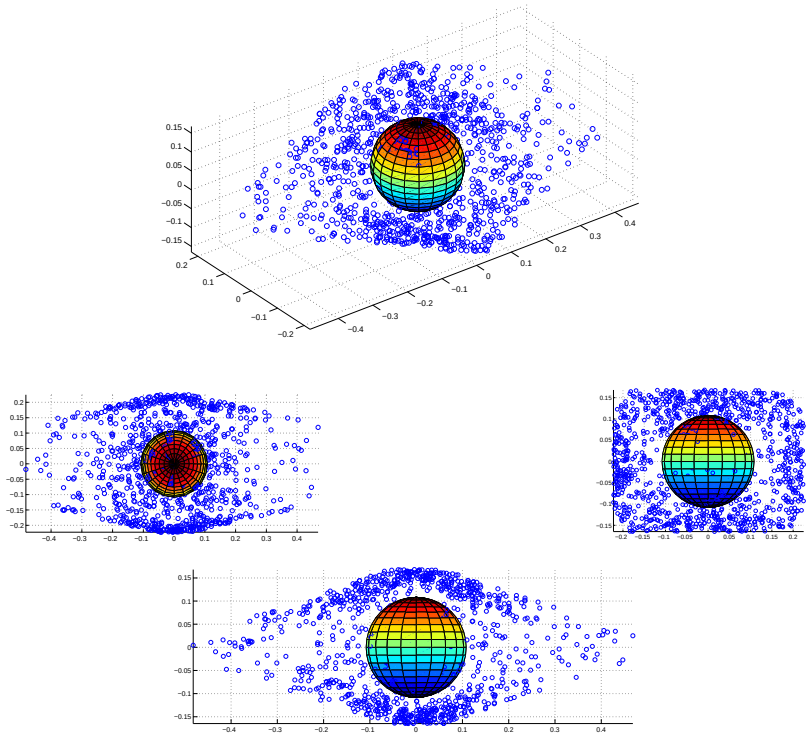


Figure 3.1: Approximation of inhomogeneous constraint (4.5) using homogeneous constraint (3.9). The maximum allowable geodesic distances for different rotation axes are shown as blue points. The homogeneous geodesic distance constraint is shown as the sphere. The bottom three figures show the same as the top figure from three perpendicular views.

In this example, the original frame size is 720×480 and the cropped

rectangle is at the center of the frame with size 540×360. For each possible rotation axis (denoted by the tangent vector $\log_{\bar{\mathbf{R}}_i} \mathbf{R}_i$ after normalization), I find the maximum geodesic distance that guarantees constraint (4.5) is satisfied. In Fig. 3.1 the homogeneous constraint is shown as the sphere and the maximum allowable geodesic distance for each rotation axis (I uniformly sample 1000 rotation axes) is shown as a blue point. I also show three perpendicular views to better illustrate the difference between the two constraints. From Fig. 3.1 we can observe that for most rotation axes the maximum allowable geodesic distance is close to the homogeneous bound.

The constraint (3.9) has two significant properties. First, it has a simple form – each rotation matrix in the sequence is constrained in a geodesic ball. As a result, the constraint set is a Cartesian product of geodesic balls. This property guarantees that gradient projection algorithms can be executed efficiently, as shown in the next section. Second, the constraint set is geodesic convex – given any two points in the set, there is a minimizing geodesic contained within the set that joins those two points (The geodesic convexity of the constraint can be easily proved by the triangular inequality of Riemannian metrics). Geodesic convexity is a natural generalization of convexity in Euclidean space to Riemannian manifolds. In the next section I will prove that the objective function (3.5) is also geodesic convex and thus global optimality can be guaranteed by the proposed optimization algorithms.

3.5 Motion Smoothing via Manifold Optimization

For brevity, I use $\mathbf{x} \in \mathcal{M}_R$ to represent the rotation matrix sequence $\{\mathbf{R}_i\}$ and write the objective function to minimize (3.5) as $f(\mathbf{x})$. In addition, I define $\mathbf{R}_i = A_i\mathbf{x}$, where A_i is a $3 \times 3N$ matrix that is used to extract $\mathbf{R}_i$ from $\mathbf{x}$. Similarly we can map $\mathbf{R}_i$ back to its corresponding location in $\mathbf{x}$ as $A_i^{\mathrm{T}}\mathbf{R}_i$.

The constrained motion smoothing can be finally formulated as

$$\min_{\mathbf{x}} f(\mathbf{x}), \text{ s.t. } \mathbf{x} \in \Omega, \tag{3.10}$$

where $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_N$ is a Cartesian product of geodesic balls. Each geodesic ball is defined as in (3.9):

$$\Omega_i = \{\mathbf{R}_i \in \mathbf{SO(3)} : || \log_{\tilde{\mathbf{R}}_i} \mathbf{R}_i||_F \leq r_0\}. \tag{3.11}$$

3.5.1 Unconstrained Optimization

In this subsection I first ignore the constraint and only minimize the objective function in the entire manifold $\mathcal{M}_R$. I consider the constrained manifold optimization in the next subsection.

As I mentioned, I will directly solve the optimization problem using manifold optimization methods. In other words, the optimization algorithms are based on the geometric structure of the manifold, not its embedding Euclidean space. In fact, the problem in (3.10) without the constraint is equivalent to an unconstrained quadratic programming problem in Euclidean space. In Euclidean space, such problems have closed-form solution; however, on nonlinear manifolds we have to use iterative algorithms.

Similar with Euclidean space, gradient-related iterative algorithms are widely used in optimization on manifolds [1]. The gradient-related algorithms for unconstrained optimization problem on the manifold $\mathcal{M}_R$ can be summarized as follows: For any element $\mathbf{x}$ in the manifold of rotation matrix sequence $\mathcal{M}_R$, given any tangent vector $\xi_{\mathbf{x}} \in T_{\mathbf{x}}\mathcal{M}_R$, we can move $\mathbf{x}$ along the direction defined by $\xi_{\mathbf{x}}$ using the exponential mapping $\exp_{\mathbf{x}}\xi_{\mathbf{x}}$. Note that given the separability property of the tangent vectors the exponential mapping can also be implemented separately for different rotation matrices in the sequence. If $\xi_{\mathbf{x}}$ is a descent direction related to the gradient of the objective function at $\mathbf{x}$, then we have the gradient-related algorithm on the manifold $\mathcal{M}_R$. In fact, similar convergence results of gradient-related algorithms has been extended from Euclidean space to any manifold [1]. The gradient-related algorithms can be classified according to the choice of the descent directions. Popular gradient-related algorithms include steepest gradient descent, conjugate gradient descent, Newton's method, etc.

In this chapter I investigate steepest gradient descent and Newton's method, which needs the computation of gradient and Hessian of the objective function.

3.5.2 Gradient Computation

In manifold, the gradient of a function is defined as follows:

Definition 1. For any real-valued function $f : \mathcal{M} \to \mathbb{R}$ defined on manifold

$\mathcal{M}$, the gradient $\mathrm{grad}f(\mathbf{x})$ is a vector field that satisfies

$$\langle \mathrm{grad}f(\mathbf{x}), \xi_{\mathbf{x}} \rangle_{\mathbf{x}} = \mathrm{D}f(\mathbf{x})[\xi_{\mathbf{x}}], \forall \xi_{\mathbf{x}} \in T_{\mathbf{x}}\mathcal{M}, \tag{3.12}$$

where $\langle \cdot, \cdot \rangle_{\mathbf{x}}$ on the left-hand side of (3.12) is any inner product in the tangent space $T_{\mathbf{x}}\mathcal{M}$ that induces a Riemannian metric, $\mathrm{D}f(\mathbf{x})[\cdot]$ on the right-hand side of (3.12) is the differential map of f at $\mathbf{x}$.

To compute the gradient we first rewrite the objective function as

$$f(\mathbf{x}) = \sum_{i=1}^{N} g_i(\mathbf{x}) + \alpha \sum_{i=1}^{N-1} h_i(\mathbf{x}), \tag{3.13}$$

where $g_i(\mathbf{x}) = \frac{1}{2} d_g^2(\tilde{\mathbf{R}}_i, \mathbf{R}_i)$ and $h_i(\mathbf{x}) = \frac{1}{2} d_g^2(\mathbf{R}_i, \mathbf{R}_{i+1})$. Note that $\mathbf{R}_i = A_i\mathbf{x}$ in our notation.

If we consider $\frac{1}{2} d_g^2(\tilde{\mathbf{R}}_i, \mathbf{R}_i)$ as a function of $\mathbf{R}_i$, it has been proved [45] that

$$\mathrm{grad}\frac{1}{2} d_g^2(\tilde{\mathbf{R}}_i, \mathbf{R}_i) = -\log_{\mathbf{R}_i} \tilde{\mathbf{R}}_i. \tag{3.14}$$

Given the separability feature of $\mathbf{x}$, we can further obtain

$$\mathrm{grad}g_i(\mathbf{x}) = -A_i^{\mathrm{T}} \log_{A_i\mathbf{x}} \tilde{\mathbf{R}}_i \tag{3.15}$$

I propose the following lemma to compute the gradient of $h_i(\mathbf{x})$:

Lemma 1. The gradient of the function $h_i(\mathbf{x})$ defined in (3.13) on manifold $\mathcal{M}_R$ can be represented as

$$\mathrm{grad}h_i(\mathbf{x}) = -A_i^{\mathrm{T}} \log_{A_i\mathbf{x}} A_{i+1}\mathbf{x} - A_{i+1}^{\mathrm{T}} \log_{A_{i+1}\mathbf{x}} A_i\mathbf{x} \tag{3.16}$$

Proof. $\forall \xi_{\mathbf{x}} \in T_{\mathbf{x}}\mathcal{M}_R$, define a geodesic curve $\gamma(t) = \exp_{\mathbf{x}}(t\xi_{\mathbf{x}})$, then from the definition of differential map we have

$$\mathrm{D}h_i(\mathbf{x})[\xi_{\mathbf{x}}] = \left.\frac{\mathrm{d}h_i(\gamma(t))}{\mathrm{d}t}\right|_{t=0}. \tag{3.17}$$

Now consider a family of geodesics

$$c(s,t) = \exp_{A_{i+1}\gamma(t)}(s \log_{A_{i+1}\gamma(t)} A_i\gamma(t)). \tag{3.18}$$

Denote

$$\begin{cases} c'(s,t) = \frac{\mathrm{d}c(s,t)}{\mathrm{d}s} \\ \dot{c}(s,t) = \frac{\mathrm{d}c(s,t)}{\mathrm{d}t}. \end{cases} \tag{3.19}$$

According to the definition of exponential mapping we have $c'(s,t) = \log_{A_{i+1}\gamma(t)} A_i\gamma(t)$ and is independent of s. Then we have

$$\begin{aligned} \frac{\mathrm{d}h_i(\gamma(t))}{\mathrm{d}t} &= \frac{\mathrm{d}}{\mathrm{d}t}\langle \log_{A_{i+1}\gamma(t)} A_i\gamma(t), \log_{A_{i+1}\gamma(t)} A_i\gamma(t)\rangle \\ &= \frac{\mathrm{d}}{\mathrm{d}t}\langle c'(s,t), c'(s,t)\rangle \\ &= \langle \frac{\mathrm{d}}{\mathrm{d}s}\dot{c}(s,t), c'(s,t)\rangle \\ &= \int_0^1 \langle \frac{\mathrm{d}}{\mathrm{d}s}\dot{c}(s,t), c'(s,t)\rangle \mathrm{d}s \\ &= \int_0^1 \frac{\mathrm{d}}{\mathrm{d}s}\langle \dot{c}(s,t), c'(s,t)\rangle \mathrm{d}s \\ &= \langle \dot{c}(1,t), c'(1,t)\rangle - \langle \dot{c}(0,t), c'(0,t)\rangle \\ &= \langle A_i\gamma'(t), \log_{A_{i+1}\gamma(t)} A_i\gamma(t)\rangle - \\ &\quad \langle A_{i+1}\gamma'(t), \log_{A_{i+1}\gamma(t)} A_i\gamma(t)\rangle. \end{aligned} \tag{3.20}$$

Note that the tangent vectors in of $\mathcal{M}_R$ also has its Cartesian product structure, so I denote $\xi_i = A_i\xi_{\mathbf{x}}$ and $\xi_{i+1} = A_{i+1}\xi_{\mathbf{x}}$. From the definition of exponential

mapping we have $\gamma'(t) = \xi_{\mathbf{x}}$. Therefore, we have

$$\begin{aligned}\left.\frac{\mathrm{d}h_i(\gamma(t))}{\mathrm{d}t}\right|_{t=0} &= \langle \xi_i - \xi_{i+1}, \log_{A_{i+1}\gamma(0)} A_i\gamma(0)\rangle \\ &= \langle \xi_i - \xi_{i+1}, \log_{A_{i+1}\mathbf{x}} A_i\mathbf{x}\rangle \\ &= \langle \xi_{\mathbf{x}}, -A_i^{\mathrm{T}} \log_{A_i\mathbf{x}} A_{i+1}\mathbf{x} - \\ &\quad A_{i+1}^{\mathrm{T}} \log_{A_{i+1}\mathbf{x}} A_i\mathbf{x}\rangle. \end{aligned} \tag{3.21}$$

According to the definition of gradient I have now proved Lemma 1. □

Now I have derived the gradient of $g_i(\mathbf{x})$ and $h_i(\mathbf{x})$. Using linearity of the gradient, we can obtain

$$\begin{aligned}\mathrm{grad}f(\mathbf{x}) = -A_1^{\mathrm{T}}(\log_{A_1\mathbf{x}}\tilde{\mathbf{R}}_1 + \log_{A_1\mathbf{x}} A_2\mathbf{x}) - \\ \sum_{i=2}^{N-1} A_i^{\mathrm{T}}(\log_{A_i\mathbf{x}}\tilde{\mathbf{R}}_i + \log_{A_i\mathbf{x}} A_{i+1}\mathbf{x} + \log_{A_i\mathbf{x}} A_{i-1}\mathbf{x}) - \\ A_N^{\mathrm{T}}(\log_{A_N\mathbf{x}}\tilde{\mathbf{R}}_N + \log_{A_N\mathbf{x}} A_{N-1}\mathbf{x}).\end{aligned} \tag{3.22}$$

Equation (3.22) clearly shows the decomposition of $\mathrm{grad}f(\mathbf{x})$ into N skew symmetric matrices corresponding to the N rotation matrices in $\mathbf{x}$. Given the direction, we can use exponential mapping to update $\mathbf{x}$ in each iteration for steepest gradient descent algorithm.

3.5.3 Hessian Computation

In Euclidean space the convergence rate of steepest gradient descent is strongly affected by the eigenvalues of the Hessian matrix of the objective function $\mathrm{Hess}f(\mathbf{x})$. This property also holds for non-linear manifolds [1].

In fact we can check that the Hessian matrix of the given objective function is ill-conditioned (the largest eigenvalue is much larger than the smallest eigenvalue). Therefore, the steepest gradient descent method converges only sublinearly.

Newton's method has been proved to converge locally quadratically to the optimal solution for both Euclidean space and non-linear manifolds. Especially for general Riemannian manifolds, the framework of Newton's method was first proposed in [16, 81] with a proof of quadratic convergence.

Newton's method needs calculating the Hessian $\mathrm{Hess} f(\mathbf{x})$. In manifolds the Hessian is defined as following:

Definition 2. For any real-valued function $f : \mathcal{M} \to \mathbb{R}$ defined on a Riemannian manifold $\mathcal{M}$ with Levi-Civita Connection $\bigtriangledown$, the Hessian $\mathrm{Hess} f(\mathbf{x})$ is mapping from $T_{\mathbf{x}}\mathcal{M}$ to $T_{\mathbf{x}}\mathcal{M}$ satisfying

$$\mathrm{Hess} f(\mathbf{x})[\xi_{\mathbf{x}}] = \bigtriangledown_{\xi_{\mathbf{x}}} \mathrm{grad} f(\mathbf{x}). \tag{3.23}$$

Note that the Levi-Civita Connection $\bigtriangledown_{\xi_{\mathbf{x}}} \mathrm{grad} f(\mathbf{x})$ is a kind of affine connection that measures in the change in $\mathrm{grad} f(\mathbf{x})$ when $\mathbf{x}$ changes infinitesimally in the direction of $\xi_{\mathbf{x}}$ [25]. The Hessian is also usually defined as an symmetric operator on two tangent vectors as

$$\mathrm{Hess} f(\mathbf{x})(\xi_{\mathbf{x}}, \eta_{\mathbf{x}}) = \langle \mathrm{Hess} f(\mathbf{x})[\xi_{\mathbf{x}}], \eta_{\mathbf{x}} \rangle = \langle \mathrm{Hess} f(\mathbf{x})[\eta_{\mathbf{x}}], \xi_{\mathbf{x}} \rangle \tag{3.24}$$

To calculate the Hessian on manifolds is a very difficult task. I start to derive the Hessian of the proposed objective function from the following lemma in [20].

Lemma 2. Consider the geodesic distance function $\phi_{\mathbf{Q}}(\mathbf{P}) = \frac{1}{2}d_g^2(\mathbf{P},\mathbf{Q})$, where $\mathbf{P},\mathbf{Q} \in \mathbf{SO(3)}$. Let $r = d_g(\mathbf{P},\mathbf{Q})$ be the geodesic distance. Let $\gamma(t) : [0,r] \rightarrow \mathbf{SO(3)}$ denote the unit speed geodesic connecting $\mathbf{Q}$ to $\mathbf{P}$. $\forall \xi_{\mathbf{P}}, \eta_{\mathbf{P}} \in T_{\mathbf{P}}\mathbf{SO(3)}$, we have the Hessian operator

$$\mathrm{Hess}\phi_{\mathbf{Q}}(\mathbf{P})(\xi_{\mathbf{P}},\eta_{\mathbf{P}}) = \langle \xi_{\mathbf{P}}^{\parallel}, \eta_{\mathbf{P}}^{\parallel} \rangle + \frac{r}{\tan(r/2)} \langle \xi_{\mathbf{P}}^{\perp}, \eta_{\mathbf{P}}^{\perp} \rangle, \tag{3.25}$$

where $\parallel$ and $\perp$ signs denote parallel and perpendicular orthogonal components of the tangent vector with respect to $\dot{\gamma}(r)$. Here $\dot{\gamma}(r) \in T_{\mathbf{P}}\mathbf{SO(3)}$ is the parallel translation of $\dot{\gamma}(0) = \log_{\mathbf{Q}}\mathbf{P}$ along the geodesic from $\mathbf{Q}$ to $\mathbf{P}$.

Given Lemma 2 and any orthonormal basis $\{E^n\}_{n=1,2,3}$ of $T_{\mathbf{P}}\mathbf{SO(3)}$ we can compute the matrix representation of the Hessian operator by computing its result on every pair of basis tangent vectors. Lemma 2 gives us a way to compute the Hessian matrix when the objective function is the geodesic distance defined on $\mathbf{SO(3)}$. In my proposed problem I need to find the Hessian for $g_i(\mathbf{x})$ and $h_i(\mathbf{x})$, which are defined on the manifold $\mathcal{M}_R$ of rotation matrix sequences. Note that due to the separability feature (Cartesian product structure) of the tangent vectors of $\mathcal{M}_R$, we can always find an orthonormal basis $\{E_i^n\}_{n=1,2,3;i=1,\ldots,N}$ of $T_{\mathbf{x}}\mathcal{M}_R$, where only $A_iE_i^n$ is non-zero and it is equal to the basis vector E^n defined for $T_{A_i\mathbf{x}}\mathbf{SO(3)}$. In other words, the orthonormal basis of $T_{\mathbf{x}}\mathcal{M}_R$ can be represented by N subgroups and each subgroup corresponds to one particular rotation matrix in the entire sequence. I propose the following proposition:

Proposition 1. Given the decomposed objective functions defined in (3.13) and an orthonormal basis of $T_{\mathbf{x}}\mathcal{M}_R$ in form of $\{E_i^n\}$, we have

$$\begin{cases} \text{Hess}g_i(\mathbf{x})(E_i^n, E_i^m) = \text{Hess}\phi_{\tilde{\mathbf{R}}_i}(A_i\mathbf{x})(E^n, E^m) \\ \text{Hess}g_i(\mathbf{x})(E_j^n, E_k^m) = 0, \text{if } j \neq i \text{ or } k \neq i \end{cases} \tag{3.26}$$

$$\begin{cases} \text{Hess}h_i(\mathbf{x})(E_i^n, E_i^m) = \text{Hess}\phi_{A_{i+1}\mathbf{x}}(A_i\mathbf{x})(E^n, E^m) \\ \text{Hess}h_i(\mathbf{x})(E_{i+1}^n, E_{i+1}^m) = \text{Hess}\phi_{A_i\mathbf{x}}(A_{i+1}\mathbf{x})(E^n, E^m) \\ \text{Hess}h_i(\mathbf{x})(E_i^n, E_{i+1}^m) = -\text{Hess}\phi_{A_{i+1}\mathbf{x}}(A_i\mathbf{x})(E^n, E^m) \\ \text{Hess}h_i(\mathbf{x})(E_{i+1}^n, E_i^m) = -\text{Hess}\phi_{A_i\mathbf{x}}(A_{i+1}\mathbf{x})(E^n, E^m) \\ \text{Hess}h_i(\mathbf{x})(E_j^n, E_k^m) = 0, \text{if } j \neq i, i+1 \text{ or } k \neq i, i+1 \end{cases} \tag{3.27}$$

The computations on the right-hand sides of (3.26) and (3.27) have been defined in Lemma 2.

Proposition 1 can be easily proved using the Cartesian product structure of $\mathcal{M}_R$ and the definitions of gradient and Hessian. Using Proposition 1 and linearity of the Hessian we can obtain a $3N \times 3N$ matrix representation H of $\text{Hess}f(\mathbf{x})$ for a given orthonormal basis $\{E_i^n\}$. To compute the direction in Newton's method, we first compute $-\text{grad}f(\mathbf{x})$ and then represent it as a vector v under the orthonormal basis $\{E_i^n\}$. Then we just need to solve the linear system $H \cdot u = v$ and the direction is represented by the vector u under the same basis.

Given any gradient-related update direction I use the Armijo rule [70] to select the step size.

3.5.4 Constrained Optimization

In Euclidean space, if the optimization problem is constrained, the update with the descent direction may be outside the constraint set. When the constraint set is convex and the update direction is the gradient, an option is to project the update onto the constraint set in each iteration. This is known as gradient projection algorithm [6] and it only works fast when the constraint set has simple form so the projection step is easy to implement, such as box constraints. The limitation of this algorithm is that the update direction can only be the gradient of the objective function. If the update direction is a scaled version of the gradient, such as in Newton's method, the projection step on the convex set should also be based on the same scaling of the original metric. This will make the projection step very hard to implement even if the constraint set has a very simple form.

In [5, 24] the authors proposed a new version of scaled gradient projection method called "two-metric projection method", which can use the scaled gradient as update direction while keep the projection step based on the original metric. The most important step in two-metric projection method is to decompose the gradient in a pair of dual cones determined by the constraint set and only scale one component. When the gradient is scaled by the Hessian (similar to Newton's method), it has been proved that the two-metric projection method can converge to a stationary point globally and has superlinear convergence rate locally around the stationary point. A clearer summary of the general form of two-metric projection method and its modification can be

found in [13, 14]. In this chapter I will first apply the two-metric projection algorithm on Euclidean space with the constraint set being a Cartesian product of Euclidean balls. Then I will extend the algorithm into optimization in manifold with the constraint set being a Cartesian product of geodesic balls. To my knowledge this is the first time that the two-metric projection method is extended to solve a constrained manifold optimization problem.

3.5.5 Two-metric method in Euclidean space

First I review the algorithm prototype proposed in [24] for generalized constrained optimization problem on a convex set in Euclidean space:

$$\min_{\mathbf{x}\in\Omega} f(\mathbf{x}), \tag{3.28}$$

where f is any smooth real function on Euclidean space and Ω is any nonempty closed convex set in Euclidean space. The Two-metric Algorithm in Euclidean space (Algorithm 1) is described below.

$\mathbf{P}_*(\cdot)$ in the algorithm means the projection of a point onto a set. To accelerate convergence the linear map $S_{\mathcal{T}}$ can be chosen based on the inverse of the Hessian of the objective function. Next I show how this algorithm work when the convex constraint set Ω is a Cartesian ball of Euclidean balls:

$$\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_N, \Omega_i = \{\mathbf{x}_i : ||\mathbf{x}_i - \mathbf{c}_i|| \leq r_0\}, \tag{3.29}$$

where $\{\mathbf{c}_i\}$ are the centers of each Euclidean ball and r_0 is the constant radius of these balls. I also assume that each Euclidean ball is with dimension K.

Algorithm 1 Two-metric Algorithm in Euclidean Space

INPUT: $\mathbf{x}^0, k = 0$
repeat
  Compute $\text{grad}f(\mathbf{x}^k)$
  $\mathbf{v}_{\mathcal{N}} = s_{\mathcal{N}}\mathbf{P}_{\mathcal{N}}(-\text{grad}f(\mathbf{x}^k))$
  $\mathbf{v}_{\mathcal{T}} = \mathbf{P}_{\mathcal{T}}S_{\mathcal{T}}\mathbf{P}_{\mathcal{N}^*}(-\text{grad}f(\mathbf{x}^k))$
  $\mathbf{v} = \mathbf{v}_{\mathcal{N}} + \mathbf{v}_{\mathcal{T}}$
  Update $\mathbf{x}^{k+1} = \mathbf{P}_{\Omega}(\mathbf{x}^k + \alpha^k\mathbf{v})$
  (α^k is the step size chosen by the Armijo Rule)
  $k = k + 1$
  where:
  $\mathcal{N} = \{\mathbf{y} : \forall \mathbf{z} \in \Omega, \langle \mathbf{y}, \mathbf{z} - \mathbf{x}^k \rangle \leq 0\}$
  (The normal cone at $\mathbf{x}^k$)
  $\mathcal{N}^* = \{\mathbf{y}^* : \forall \mathbf{y} \in \mathcal{N}, \langle \mathbf{y}^*, \mathbf{y} \rangle \leq 0\}$
  (The dual cone of $\mathcal{N}$)
  $\mathcal{T} = \{\mathbf{v}_{\mathcal{N}}\}^{\perp} \cap \mathcal{N}^*$
  $[\mathcal{T}]$ = the closed linear hull of $\mathcal{T}$
  $S_{\mathcal{T}}$ = a bounded linear map from $[\mathcal{T}]$ into $[\mathcal{T}]$
  $s_{\mathcal{N}}$ = a scalar
until Convergence

According to the Cartesian product structure of Ω, we can also decompose the sets $\mathcal{N}, \mathcal{N}^*, \mathcal{T}$ in Algorithm 1 into such kind of Cartesian product form. For any $\mathbf{x} \in \Omega$, define the active index set $I(\mathbf{x}) = \{i : ||\mathbf{x}_i - \mathbf{c}_i|| = r_0\}$, then clearly we have each component of the normal cone $\mathcal{N}$ at $\mathbf{x}$ as

$$\mathcal{N}_i = \begin{cases} \{\mathbf{0}\}, & i \notin I(\mathbf{x}) \\ \{\mathbf{v}_i : \mathbf{v}_i = \lambda(\mathbf{x}_i - \mathbf{c}_i), \lambda \geq 0\}, & i \in I(\mathbf{x}) \end{cases} \tag{3.30}$$

Then we can get each component of the dual cone $\mathcal{N}^*$ as

$$\mathcal{N}_i^* = \begin{cases} \mathbb{R}^K, & i \notin I(\mathbf{x}) \\ \{\mathbf{v}_i : \langle \mathbf{v}_i, \mathbf{x}_i - \mathbf{c}_i \rangle \leq 0\}, & i \in I(\mathbf{x}) \end{cases} \tag{3.31}$$

If we define a second active index set as

$$\hat{I}(\mathbf{x}) = \{i : i \in I(\mathbf{x}), \text{and} \langle (\text{grad} f(\mathbf{x}))_i, \mathbf{x}_i - \mathbf{c}_i \rangle < 0\}, \tag{3.32}$$

then we can get each component of the projection of the gradient on to the normal cone $\mathcal{N}$ as

$$(\mathbf{v}_{\mathcal{N}})_i = \begin{cases} \mathbf{0}, & i \notin \hat{I}(\mathbf{x}) \\ -s_{\mathcal{N}} \frac{\langle (\text{grad} f(\mathbf{x}))_i, \mathbf{x}_i - \mathbf{c}_i \rangle}{||\mathbf{x}_i - \mathbf{c}_i||^2} (\mathbf{x}_i - \mathbf{c}_i), & i \in \hat{I}(\mathbf{x}) \end{cases} \tag{3.33}$$

Finally each component of the set $\mathcal{T}$ and its linear hull $[\mathcal{T}]$ can be represented as

$$\mathcal{T}_i = \begin{cases} \mathbb{R}^K, & i \notin I(\mathbf{x}) \\ \{\mathbf{v}_i : \langle \mathbf{v}_i, \mathbf{x}_i - \mathbf{c}_i \rangle \leq 0\}, & i \in I(\mathbf{x}), i \notin \hat{I}(\mathbf{x}) \\ \{\mathbf{v}_i : \mathbf{v}_i \perp (\mathbf{x}_i - \mathbf{c}_i)\}, & i \in \hat{I}(\mathbf{x}) \end{cases} \tag{3.34}$$

$$[\mathcal{T}]_i = \begin{cases} \mathbb{R}^K, & i \notin \hat{I}(\mathbf{x}) \\ \{\mathbf{v}_i : \mathbf{v}_i \perp (\mathbf{x}_i - \mathbf{c}_i)\}, & i \in \hat{I}(\mathbf{x}) \end{cases} \tag{3.35}$$

Now I construct the linear map $S_{\mathcal{T}}$. Assume the Hessian matrix of the objective function at $\mathbf{x}$ is H. Without loss of generality, by relabeling the coordinates

of $\mathbf{x}$ if necessary, I assume that there exists an index q such that $\hat{I}(\mathbf{x}) = \{q+1, q+2, \ldots, N\}$. We diagonalize the Hessian matrix with respect to $\hat{I}(\mathbf{x})$ as

$$\hat{H} = \begin{bmatrix} \tilde{H} & \mathbf{0} \\ \mathbf{0} & I \end{bmatrix}, \tag{3.36}$$

where $\tilde{H}$ is the same as the part of H corresponding to $i \notin \hat{I}(\mathbf{x})$, I is the identity matrix corresponding to $\hat{I}(\mathbf{x})$ (the last $(N-q)$ components). We define the matrix representation of the map $S_{\mathcal{T}}$ as $\hat{H}^{-1}$. It is easy to verify that $S_{\mathcal{T}}$ maps any point in $\mathcal{T}$ into $\mathcal{T}$ itself. Thus we can get each component of $\mathbf{v}_{\mathcal{T}}$ as

$$(\mathbf{v}_{\mathcal{T}})_i = \begin{cases} -(\hat{H}^{-1}\mathrm{grad}f(\mathbf{x}))_i, & i \notin I(\mathbf{x}) \\ -\mathbf{P}_{\mathcal{T}_i}(\hat{H}^{-1}\mathrm{grad}f(\mathbf{x}))_i, & i \in I(\mathbf{x}), i \notin \hat{I}(\mathbf{x}) \\ -(\mathrm{grad}f(\mathbf{x}))_i + \frac{\langle (\mathrm{grad}f(\mathbf{x}))_i, \mathbf{x}_i - \mathbf{c}_i \rangle}{||\mathbf{x}_i - \mathbf{c}_i||^2}(\mathbf{x}_i - \mathbf{c}_i), & \\ & i \in \hat{I}(\mathbf{x}) \end{cases} \tag{3.37}$$

If we choose $s_{\mathcal{N}} = 1$ then finally we can get

$$\mathbf{v}_i = \begin{cases} -(\hat{H}^{-1}\mathrm{grad}f(\mathbf{x}))_i, & i \notin I(\mathbf{x}) \\ -\mathbf{P}_{\mathcal{T}_i}(\hat{H}^{-1}\mathrm{grad}f(\mathbf{x}))_i, & i \in I(\mathbf{x}), i \notin \hat{I}(\mathbf{x}) \\ -(\mathrm{grad}f(\mathbf{x}))_i, & i \in \hat{I}(\mathbf{x}) \end{cases} \tag{3.38}$$

We can find that only for the indices not in the active set $\hat{I}(\mathbf{x})$ the gradient is scaled by the inverse of the Hessian to get the update direction. In extreme case that the active set is always empty (the boundary of the constraint set is never reached), this algorithms turns into regular Newton's method. Note that due to the Cartesian product structure all the computation except the inverse of the Hessian matrix can be performed independently for each component.

3.5.6 Scaled Gradient Projection on Manifold

Now I generalize the two-metric projection algorithm to manifolds. The convex constraint set Ω now becomes the Cartesian of geodesic balls defined in (3.9). The centers of these balls are the original camera rotation matrices $\{\tilde{\mathbf{R}}_i\}$. Based on the discussion in the previous subsection I propose the Manifold Two-metric Algorithm for Constrained Motion Smoothing (Algorithm 2) below to solve the problem in (3.10).

3.5.7 Geodesic-convexity of Motion Smoothing

In this subsection I will show that the proposed problem in (3.10) is geodesic convex on the manifold $\mathcal{M}_R$. Geodesic-convexity is a natural generalization of convexity in Euclidean space to manifolds. Similar with Euclidean space, for geodesic-convex optimization problems, local minimum is also global minimum [88]. I mentioned in Section 3.4 that the constraint set is geodesic convex according to triangular inequality of Riemannian metrics. Here I will focus on the proof of the geodesic convexity of the objective function in (3.13). From the property that a linear combination of geodesic convex functions is still geodesic convex, it is sufficient to prove the functions $g_i(\mathbf{x})$ and $h_i(\mathbf{x})$ are geodesic convex. It has been shown that the geodesic distance function $\phi_{\mathbf{Q}}(\mathbf{P}) = \frac{1}{2}d_g^2(\mathbf{P}, \mathbf{Q})$ is geodesic convex with respect to $\mathbf{P}$ inside a geodesic ball centered at $\mathbf{Q}$ with radius less than $\pi/2\sqrt{\Delta}$, where Δ is the upper bound of the sectional curvature of the manifold that $\mathbf{P}$ and $\mathbf{Q}$ lie in [45]. For manifold $\mathbf{SO(3)}$ the sectional curvature is $1/4$ everywhere. So the radius of the geodesic

Algorithm 2 Manifold Two-metric Algorithm for Constrained Motion Smoothing

INPUT: A fixed orthonormal basis $\{E_i^n\}_{n=1,2,3;i=1,\ldots,N}$ of $T_{\mathbf{x}}\mathcal{M}_R$, $\mathbf{x}^0, k = 0$

repeat

Compute $\text{grad}f(\mathbf{x}^k)$ and its representation vector $\mathbf{u}$ in the fixed basis

Compute the representation matrix H of $\text{Hess}f(\mathbf{x}^k)$ in the fixed basis

Find active set $I(\mathbf{x}^k) = \{i : ||\log_{\tilde{\mathbf{R}}_i} A_i\mathbf{x}^k|| = r_0\}$

Find active set
$\hat{I}(\mathbf{x}^k) = \{i : i \in I(\mathbf{x}), \langle(\text{grad}f(\mathbf{x}^k))_i, \log_{\tilde{\mathbf{R}}_i} A_i\mathbf{x}^k\rangle < 0\}$

Diagonalize H to $\hat{H}$ with respect to $\hat{I}(\mathbf{x}^k)$ as in (3.36)

Compute the vector representation $\mathbf{v}$ of the update direction as

$$\mathbf{v}_i = \begin{cases} -(\hat{H}^{-1}\mathbf{u})_i, & i \notin I(\mathbf{x}) \\ -\mathbf{P}_{\mathcal{T}_i}(\hat{H}^{-1}\mathbf{u})_i, & i \in I(\mathbf{x}), i \notin \hat{I}(\mathbf{x}) \\ -\mathbf{u}_i, & i \in \hat{I}(\mathbf{x}) \end{cases} \tag{3.39}$$

Compute the update direction $\mathbf{d} \in T_{\mathbf{x}^k}\mathcal{M}_R$ based on its vector representation $\mathbf{v}$

Update $\mathbf{x}^{k+1} = \mathbf{P}_\Omega(\exp_{\mathbf{x}^k} \alpha^k\mathbf{d})$
(α^k is the step size chosen by the Armijo Rule)

$k = k + 1$

where:

$\mathcal{T}_i = \{\xi_i \in T_{A_i\mathbf{x}^k}\mathbf{SO(3)} : \langle\xi_i, \log_{\tilde{\mathbf{R}}_i} A_i\mathbf{x}^k\rangle \leq 0\}, i \in I(\mathbf{x}), i \notin \hat{I}(\mathbf{x})$

until Convergence

ball centered at $\mathbf{Q}$ should be less than $\pi/2$. Consider $\mathbf{P}$ as a rotation matrix for a certain frame and the $\mathbf{Q}$ as the rotation matrix of its adjacent frame, $\mathbf{P}$ is well guaranteed to be inside the geodesic ball centered at $\mathbf{Q}$ with radius $\pi/2$ since the change of rotation between two consecutive frames is very small. If $\mathbf{Q}$ is the original rotation matrix for a certain frame and $\mathbf{P}$ is the new rotation matrix in the stabilized video for the same frame, then $\mathbf{P}$ is still guaranteed to be inside the geodesic ball centered at $\mathbf{Q}$ with radius $\pi/2$ given the proposed constraint (3.11).

It has been shown that a function is geodesic convex if and only if the matrix representation of its Hessian is positive semi-definite (PSD) [88]. I have mentioned that the function $\phi_{\tilde{\mathbf{R}}_i}(A_i\mathbf{x})$ is geodesic convex with respect to $A_i\mathbf{x}$. So the matrix representation of its Hessian matrix H_g is PSD. According to Proposition 1 the matrix representation of the Hessian of $g_i(\mathbf{x})$ can be represented as

$$\begin{bmatrix} \mathbf{0} & & & & \\ & \ddots & & & \\ & & H_g & & \\ & & & \ddots & \\ & & & & \mathbf{0} \end{bmatrix} \tag{3.40}$$

(H_g locates at the coordinates corresponding to frame i) and is clearly also PSD.

Since I have also mentioned that the function $\phi_{A_i\mathbf{x}}(A_{i+1}\mathbf{x})$ is geodesic convex with respect to $A_{i+1}\mathbf{x}$, its Hessian matrix representation H_h is PSD. From Lemma 2 the Hessian matrix representation of the function $\phi_{A_{i+1}\mathbf{x}}(A_i\mathbf{x})$ is also H_h. Thus according to Proposition 1 the Hessian matrix representation

of $h_i(\mathbf{x})$ is

$$\begin{bmatrix} \mathbf{0} & & & & & \\ & \ddots & & & & \\ & & H_h & -H_h & & \\ & & -H_h & H_h & & \\ & & & & \ddots & \\ & & & & & \mathbf{0} \end{bmatrix}. \tag{3.41}$$

It is not hard to show that this matrix is also PSD.

Therefore, I have show that the function $g_i(\mathbf{x})$ and $h_i(\mathbf{x})$ in (3.13) are geodesic convex, which means the objective function $f(\mathbf{x})$ is geodesic convex. Given the property that the proposed two-metric scaled gradient projection algorithm can always converge to a stationary point, the global optimality of the proposed algorithm is guaranteed.

3.5.8 Choice of Regularization Parameter

As I mentioned, the original objective function (3.5) consists of two different terms: data-fitting term and the regularization term. The parameter α controls the relative weights of the two different terms. The data-fitting term was originally introduced into the unconstrained video stabilization problem to guarantee that the smoothed camera motion trajectory does not deviate from the original trajectory too much. However, we can find that such requirement is redundant if the hard constraint (3.9) is added. Therefore, the optimal choice of α should be infinity in order to reach the greatest degree of smoothness in the stabilized trajectory. This is equivalent to dropping the data-fitting term $\sum_{i=1}^{N} g_i(\mathbf{x})$.

However, such setting is impractical since the proposed algorithm relies on inverting the matrix representation of $\mathrm{Hess} f(\mathbf{x})$. According to the discussion in Section 3.5.7, the Hessian matrix representation of $\sum_{i=1}^{N-1} h_i(\mathbf{x})$ can be written as

$$H = \begin{bmatrix} H_h & -H_h & & & & \\ -H_h & 2H_h & -H_h & & & \\ & -H_h & 2H_h & & & \\ & & & \ddots & & \\ & & & & 2H_h & -H_h \\ & & & & -H_h & H_h \end{bmatrix}. \tag{3.42}$$

Recall that H is a $3N \times 3N$ matrix given an orthonormal basis of $T_{\mathbf{x}}\mathcal{M}_R$. From the fact that each 3×3 matrix block H_h is PSD, we can decompose it as $H_h = {\Theta_h}^{\mathrm{T}}\Theta_h$. As a result, we can decompose H as $H = \Theta^{\mathrm{T}}\Theta$, where Θ is a $3(N-1) \times 3N$ matrix and

$$\Theta = \begin{bmatrix} \Theta_h & -\Theta_h & & & \\ & \Theta_h & -\Theta_h & & \\ & & & \ddots & \\ & & & \Theta_h & -\Theta_h \end{bmatrix}. \tag{3.43}$$

Clearly the matrix H does not have full rank and thus is not invertible. Therefore, in practice I leave the data-fitting term in the objective function and set α to a very large number to avoid the numerical problem of matrix inversion.

3.6 Experimental Results

I first compare the convergence rate of different algorithms in solving the formulated smoothing problem on the sequences of rotation matrices. Fig. 3.2 is an example showing the convergence rate of steepest gradient descent method and Newton's method in solving the unconstrained formulated

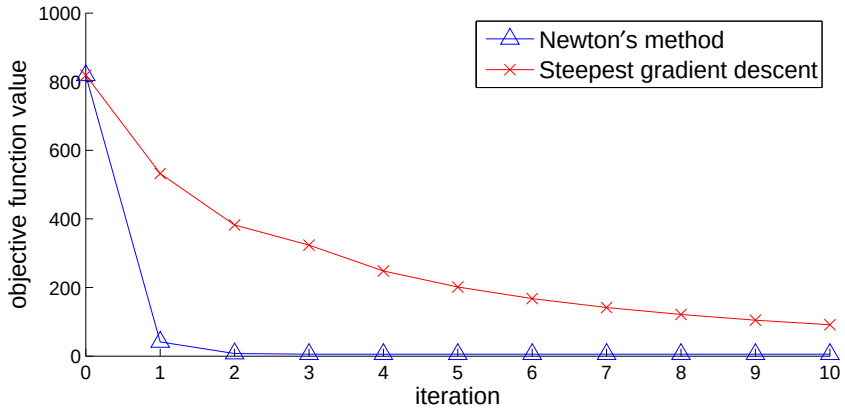


Figure 3.2: Convergence of two gradient-related algorithms for unconstrained motion smoothing: Newton's method and steepest gradient descent. For these algorithms, I compute the gradient and Hessian of the objective function using Riemannian geometry.

problem. In the experiment we try to smooth a sequence of 478 3D rotation matrices (478 frames) with $\alpha = 1000$, which were taken at a frame rate of 30 Hz. Figure 3.2 shows the values of the objective function in 10 iterations. Newton's method successfully converges in just 2 iterations. Each iteration of the Newton's method takes 2.93s on a 2.3GHz Intel i5 processor machine with MATLAB implementation (without parallel processing). From Fig. 3.2 we can observe that the scaling of gradient using Hessian matrix can significantly accelerate the convergence, which motivates us to do the same thing in solving the constrained problem.

In Fig. 3.3 I compare the convergence rate of gradient projection method and the proposed scaled gradient projection method in solving the constrained motion smoothing problem. The test video sequence is the same

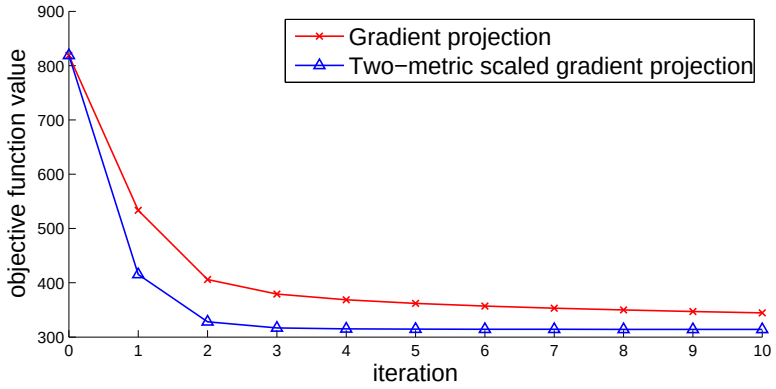


Figure 3.3: Convergence of two algorithms for constrained motion smoothing: manifold gradient projection and the manifold two-metric scaled gradient projection (proposed).

as in Fig. 3.2. The original frame size is 720×480 and the guaranteed cropped size is 540×360. The radius r_0 is found as 0.11. We can observe that proposed two-metric scaled gradient projection method successfully converges in only 4 iterations. Note that the difference in the final convergence values of the objective function between Fig. 3.2 and Fig. 3.3 is caused by the constraint in (3.9). Each iteration of the proposed method takes 3.58s on the same machine as in Section 3.5.3, while the gradient projection method takes 18.66s per iteration. The reason gradient projection method takes such longer time in each iteration is that it needs multiple tries to find the proper (descent) step size using the Armijo step size selection method. For the proposed method usually 1 is the proper step size (no more tries are needed) because the gradient has been scaled by the Hessian of the objective function.

Fig. 3.4 shows another example comparing the convergence rate of gra-

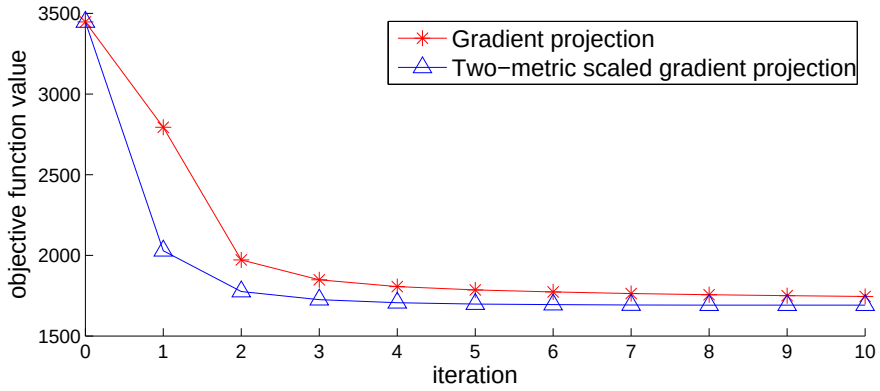


Figure 3.4: Convergence of two algorithms for constrained motion smoothing: gradient projection and the manifold two-metric scaled gradient projection (proposed).

dient projection method and the proposed scaled gradient projection method. In this example there are 761 frames in the test video sequence. The proposed two-metric scaled gradient projection method successfully converges in only 5 iterations. In Fig. 3.5 I additionally run the proposed two-metric projection method on a video with 163 frames and show the running time of each iteration for all of the three test videos with respect to their numbers of frames. We can observe that the running time of each iteration increases linearly with the increase in the number of frames, thanks to the sparse structure of the Hessian matrix as shown in Proposition 1.

A comparison between the original camera rotation and the smoothed camera rotation is shown in Fig. 3.6. In this figure, the camera rotation corresponding to each frame in the video sequence is shown in form of Tait-Bryan angles (a similar representation as Euler angles). Note that, however,

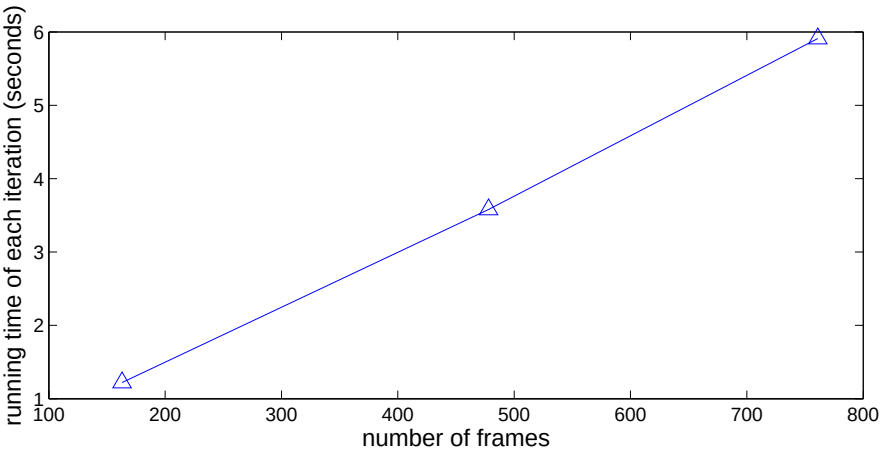


Figure 3.5: Running time of each iteration for the proposed manifold two-metric projection algorithm.

our motion smoothing is performed naturally on the manifold $\mathcal{M}_R$ instead of the Euclidean space of rotation angle representation.

Then I use the proposed constrained motion smoothing method in video stabilization. In the experiments I try to stabilize the video sequences captured by Google Nexus S smart phone. While recording videos, I also captured the readings (with timestamps) from the 3-axis gyroscope inside the phone. The camera has been calibrated so the camera intrinsic matrix $\mathbf{K}$ is known. I also assume that the gyroscope and the videos have been synchronized so that we can obtain the camera pose (3D rotation) simply from the gyroscope readings. In practice, the calibration and synchronization can be executed using the method provided in Chapter 2.

I use the proposed constrained motion smoothing method in video stabilization and compare my method with the YouTube video editor. The video

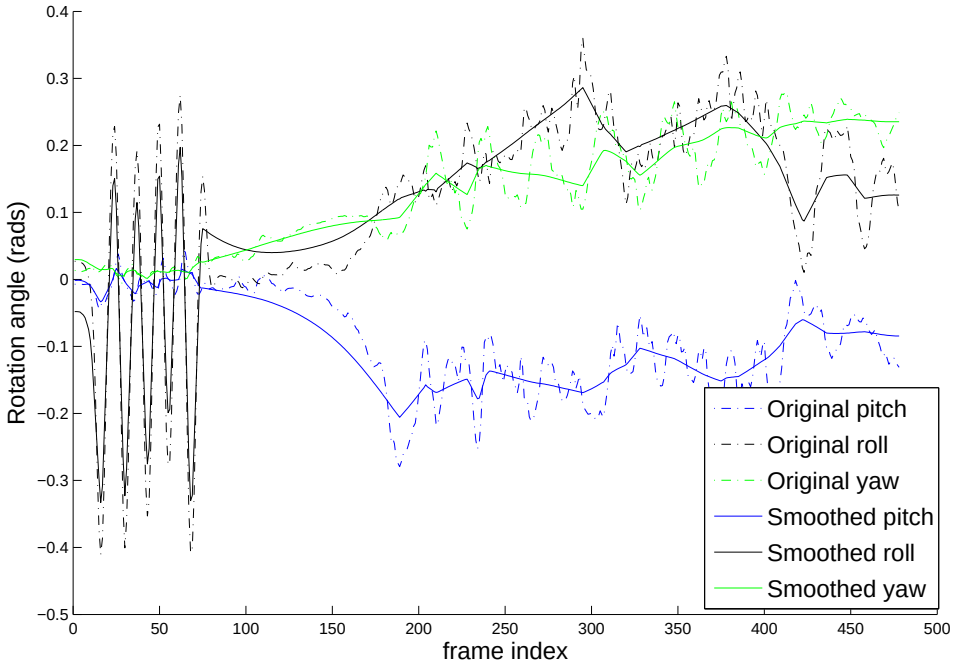


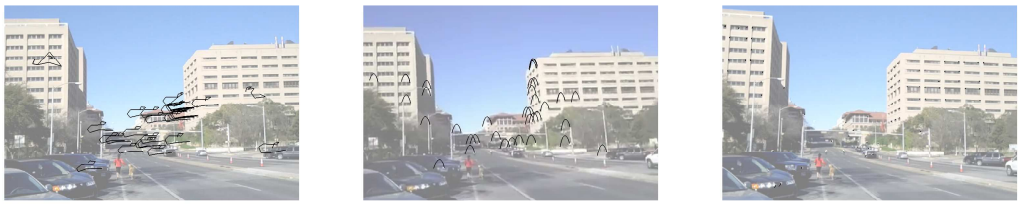
Figure 3.6: Comparison of the original and the smoothed camera rotation using the proposed two-metric scaled gradient projection method for constrained smoothing on the manifold of rotation matrices.

stabilization in YouTube video editor is based on the approach proposed in [29], which estimates the 2D similarity camera motion (with homography refinement) from frame to frame and uses L^1 regularization to smooth the estimated camera path. This method is one of the state-of-the-art video stabilization algorithms and may be the only one that explicitly considers the constraints for black borders in motion smoothing. I compared the algorithms on videos with original size 720×480. The cropping size used in YouTube video editor is 540×360 so I use the same size in our method. In all of the

experiments I fix the smoothness parameter in our approach as $\alpha = 1000$ and find very little difference in the results with higher value of α.

First, I test the video stabilization algorithms on a video shot by a walking forward person. I use feature trajectories as shown in Fig. 3.7 as a visualization of the changing frames. I detect Harris corner points in a certain frame (no. 270 in Fig. 3.7) and track them for ten frames. The feature trajectories are plotted as black curves on top of the starting frame (the frames themselves are plotted using alpha channel 0.5 (more transparent) to make the curves clearer). For a stabilized video the trajectories should be very short since the camera is always facing forward in spite of jitter caused by camera shake. The 2D L^1 regularization method [29] can smooth and shorten the trajectories compared to the original video, but the feature points are still moving up and down. Our algorithm can keep the feature points very steady and the trajectories become almost invisible (just black dots) in the results. Note that I detect and track the feature points independently in the three videos so the location and number of the feature points can be different.

Next I take a test on a video shot while panning the camera. Video stabilization should only remove the unwanted jitter while keeping the panning motion of the camera. In Fig. 3.8 I do the same test as in Fig. 3.7, except that the feature points are tracked in twenty consecutive frames instead of ten. Both methods successfully smooth the trajectories of the feature points. However, the trajectories in the stabilization result of [29] is not as straight as those in the result of the proposed method. This comparison, together

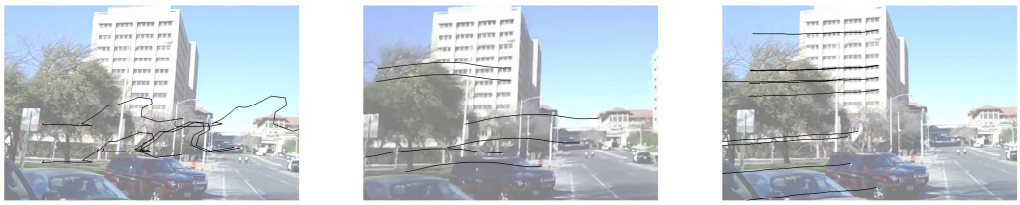


(a) Original video (b) YouTube video editor [29] (c) Proposed method

Figure 3.7: Stabilization comparison for a video shot by a walking forward person. Features are tracked from frame 270 to frame 280. The feature trajectories are plotted as black curves on frame 270.

with the comparison in Fig. 3.7, show that the proposed method can not only better remove high frequency unwanted jitter but also better smooth the long term motion of the camera.

The stabilization results are best viewed in video form. Please see the online video examples at [37]. The results of YouTube video editor have been compressed so please ignore these compression artifacts in comparison. Besides compression, please note that there is some non-rigid wobble in the results of YouTube video editor. This is not caused by compression but the inaccuracy of the 2D motion model used in [29]. The 3D rotational model in my video stabilization accurately reflect the real camera motion so there is no such non-rigid distortion. In the two video examples features are easy to track since there is very little motion blur in the frames. However, when the videos are shot in low light condition the visual-based motion estimation used in [29] will fail sometimes while 3D rotational video stabilization using gyroscopes is



(a) Original video (b) YouTube video editor [29] (c) Proposed method

Figure 3.8: Stabilization comparison for a video shot while panning the camera. Features are tracked from frame 650 to frame 670. The feature tracks are plotted as black curves on frame 650.

not affected.

As I mentioned in Section 3.2, other 3D rotational video stabilization algorithms [31, 46] are based on local low-pass filtering of the rotation sequence and thus are not able to guarantee that there will be no black borders. Many frames in the stabilized video still have black borders even though adaptive filtering with different window size could be applied to decrease their area. In [31], the authors proposed to use extrapolation to fill the black borders. Extrapolation can be implemented very fast but the image quality is severely sacrificed, as shown in Fig. 3.9.

3.7 Conclusions

In this chapter I propose a novel video stabilization method using a 3D rotational camera motion model. I exploit the manifold structure of not only the 3D rotation matrices, but also the sequences of 3D rotation matrices. This

Figure 3.9: Extrapolation is used to fill the undefined areas (black borders) on the left and top of the frame [31].

allows us to globally formulate motion smoothing as a regression problem based on geodesic distance. Furthermore, I force the solution to lie on a Cartesian product of geodesic balls so that every pixel in the stabilized frame is visible in the original frame. I directly solve the formulated problem on manifold by generalizing the existing two-metric projection algorithm in Euclidean space. The 3D camera rotation for each frame is obtained reliably using gyroscopes that are equipped in most smart phones and tablets, no matter whether there is motion blur or abrupt illumination change in the videos. I have demonstrated in experiments that our algorithm is very fast and can generate better video stabilization results than state-of-the-art methods.

Chapter 4

Real-time 3D Rotation Smoothing

4.1 Introduction

This chapter focuses on real-time (online) motion smoothing.

Given the estimated camera motion for each frame, motion smoothing aims at designing a new smooth camera motion path. Most existing works address motion smoothing as an offline processing after the entire video sequence has been recorded. However, real-time video stabilization is necessary for applications such as video conferencing and broadcasting. Besides, for consumers who want to record videos, real-time stabilization can greatly improve the user experience with the stabilized videos displayed in real-time on the viewfinders. Real-time video stabilization is also able to reduce the memory requirements with frames stabilized before compression. In real-time video stabilization, camera motion is required to be smoothed in a causal way. This is more difficult than offline motion smoothing because we are missing information of how camera motion changes afterward.

In this chapter, I focus on real-time motion smoothing based on a 3D rotational camera motion model for a calibrated camera with a known intrinsic matrix. Compared to 2D (translational, Euclidean, or affine) motion models,

3D motion models can reflect the real camera projection more accurately, and thus give more realistic motion smoothing and avoid image distortion in frame synthesis. I ignore 3D translation of the camera because (1) the unwanted jitter in videos are primarily caused by camera rotation, and (2) frame synthesis with 3D camera translation would need the depth value at every pixel, which is very difficult to obtain accurately. To estimate the 3D camera rotation we use a gyroscope that is available in many smart phones and tablets. Current gyroscopes in smart phones have very high precision and can return more reliable 3D camera rotation estimates compared to the estimates obtained from visual features in the video sequence, especially when there are many moving objects in the scene or it is difficult to track feature points due to motion blur or illumination changes.

Under a 3D rotational model, camera motion for a video can be considered as a sequence of 3D rotation matrices. I propose two algorithms for real-time 3D rotation smoothing, generalized from two classical 2D motion smoothing algorithms. The first algorithm smooths the 3D rotation sequence similar to 1st-order IIR filtering. The second algorithm works as sequential estimation with a constant angular velocity model. I exploit the manifold structure of the rotation matrices so that the proposed algorithms directly smooth the 3D rotation sequences on the manifold.

Due to the camera motion change from motion smoothing, some areas in the synthesized frame will be undefined. This is known as black border problem. In practice we have to crop the resulting video frames and enlarge

them if necessary. Still, in motion smoothing, we have to constrain the change of camera motion in order to guarantee that no black borders intrude into the stabilized video frames. This is achieved by adding a projection step at each frame.

This chapter is organized as follows: Section 4.2 reviews previous motion smoothing algorithms. Section 4.3 briefly introduces the manifold of 3D rotation matrices. Section 4.4 shows how black-border constraint can be modeled on the manifold and how the projection step works. The two proposed real-time rotation smoothing algorithms are presented in Section 4.5 and Section 4.6, respectively. Section 4.7 shows how the proposed rotation smoothing algorithms can effectively stabilize the video sequences. Section 4.8 concludes the chapter.

4.2 Related Work

Many (if not most) existing motion smoothing algorithms are offline smoothing. Gaussian window filtering was used to smooth the entire camera motion path in [19, 62] under 2D translational and affine model respectively. Another kind of algorithms smooth the camera motion via minimizing a certain objective function that represents the smoothness of the camera motion trajectory. An advantage of such objective-minimizing methods is that the black-border constraints can be naturally added to the problem and solved by constrained optimization. In [82], the authors defined the objective function as the L^2 norm of the second order difference of camera motion under 2D

Euclidean model. The black border constraint was approximately modeled by an interval constraint on the motion parameters. Similar modeling was also used in [73], but the variables were assumed to be integer-valued and the problem was solved via dynamic programming. In [29] the objective function was a mixture of the first, second and third order difference of camera motion measured by L^1 norm. The motion model was 2D similarity motion and the black-border constraint was modeled precisely as linear inequalities. As a result, the constrained motion smoothing could be solved efficiently by linear programming. Black-border constrained was also taken into consideration for window-filtering-based methods. In [72] the authors proposed a dual pass motion smoothing method which could find an optimal cropping size as large as possible.

In [17] IIR filtering was proposed for online motion smoothing based on 2D translational motion model. Kalman filtering was first used for online smoothing in [18]. The intentional motion parameters (under 2D translational motion model) were modeled by a constant velocity linear system so Kalman filtering could be used to optimally estimate them. The same Kalman-filtering motion smoothing framework was extended to 2D affine motion model in [55], leading a better performance. The same algorithm was widely used in the later video stabilization works, such as [92].

The black-border constraints were rarely considered in online motion smoothing. In [86] the authors proposed to use constrained Kalman filtering for 2D translational motion model. Because of the simplicity of the motion

model, interval constraints could be used and the constrained estimate could be obtained in one step. The constrained Kalman filtering is solved by estimate projection as proposed in [79]. For a more comprehensive survey of constrained Kalman filtering algorithms please see [78].

All of the above methods use 2D camera motion models. Using full 3D models including both rotation and translation for calibrated cameras was first proposed in [8] and further discussed in [56]. In both papers complicated approximations are used in frame synthesis to handle the problem of missing depth values. In [46, 65] pure 3D rotational models with known intrinsic camera parameters were shown to generate high-quality results while only needing homography-based warping in frame synthesis.

So far to my knowledge, all of the existing 3D rotation smoothing algorithms are offline processing. Local low-pass filtering based on geodesic distance on manifold was used in [31, 56]. In [39], offline rotation smoothing was formulated globally as a regression problem on the manifold.

4.3 3D Rotation and Geodesic Distance

All of the 3×3 rotation matrices constitute the Special Orthogonal Group $\mathbf{SO(3)}$, in which any element $\mathbf{R}$ satisfies the constraint $\mathbf{RR}^{\mathrm{T}} = \mathbf{I}$. $\mathbf{SO(3)}$ can be also considered as an embedded Riemannian submanifold of Euclidean space $\mathbb{R}^9$ (represented as 3×3 real matrices). A natural extension of Euclidean distance in Euclidean space to the Riemannian manifold $\mathbf{SO(3)}$

is the geodesic distance

$$d_g(\mathbf{R}_i, \mathbf{R}_j) = ||\text{logm}(\mathbf{R}_i^{\text{T}}\mathbf{R}_j)||_F, \tag{4.1}$$

where $\text{logm}(\cdot)$ is the matrix logarithm operator and $||\cdot||_F$ is the Frobenius norm of a matrix. In fact, $\text{logm}(\mathbf{R}_i^{\text{T}}\mathbf{R}_j)$ is a skew-symmetric matrix representing a tangent vector in the tangent space $T_{\mathbf{R}_i}\mathbf{SO}(\mathbf{3})$ that indicates the non-normalized direction from $\mathbf{R}_i$ to $\mathbf{R}_j$ on $\mathbf{SO}(\mathbf{3})$. Usually we also write $\text{logm}(\mathbf{R}_i'\mathbf{R}_j)$ as $\log_{\mathbf{R}_i}\mathbf{R}_j$ and call it the logarithmic mapping. Inversely, given any tangent vector $\xi \in T_{\mathbf{R}_i}\mathbf{SO}(\mathbf{3})$, we can define $\exp_{\mathbf{R}_i}\xi = \mathbf{R}_i\text{expm}(\xi)$, where $\text{expm}(\cdot)$ is the matrix exponential operator. Here, $\exp_{\mathbf{R}_i}\xi$ is called the exponential mapping and is used to move $\mathbf{R}_i$ along the direction defined by ξ on $\mathbf{SO}(\mathbf{3})$. The logarithmic mapping and exponential mapping together define a curve

$$t \in [0,1] \mapsto \gamma(t) = \exp_{\mathbf{R}_i}\left(t \cdot \log_{\mathbf{R}_i}\mathbf{R}_j\right), \tag{4.2}$$

which is known as the minimizing geodesic from $\mathbf{R}_i$ to $\mathbf{R}_j$ on $\mathbf{SO}(\mathbf{3})$. The minimizing geodesic is a generalization of the notion of "straight line" in Euclidean space to Riemannian manifolds, representing the shortest path between two points in the manifolds given a Riemannian metric. The length of the minimizing geodesic is defined in (4.1).

4.4 Black-Border Constraint and Estimate Projection

In the last step of video stabilization, the synthesized frames may contain black borders since not every pixel in the synthesized frame is visible in

the original frame due to the change of camera orientation. Therefore, we have to crop the synthesized frames into a smaller size so that there are no black borders in the stabilized video. In other words, given a preferred stabilized size of the video, the video stabilization system must guarantee that every pixel in the cropped stabilized frames is visible in the original frames. This is a hard constraint that has to be considered in the camera motion smoothing algorithm.

Assume the intrinsic projection matrix of the camera is given as $\mathbf{K}$. Under pure 3D camera rotation, for any pixel $[u_{kj}, v_{kj}]^{\mathrm{T}}$ in the stabilized frame k, its corresponding 2D pixel location in the original frame $[\tilde{u}_{kj}, \tilde{v}_{kj}]^{\mathrm{T}}$ can be computed as

$$\begin{bmatrix} \tilde{u}_{kj} \\ \tilde{v}_{kj} \end{bmatrix} = g\left(\mathbf{K}\mathbf{R}_k\hat{\mathbf{R}}_k^{-1}\mathbf{K}^{-1} \begin{bmatrix} u_{kj} \\ v_{kj} \\ 1 \end{bmatrix} \right), \tag{4.3}$$

where the function

$$g([x, y, z]^{\mathrm{T}}) = [x/z, y/z]^{\mathrm{T}} \tag{4.4}$$

is used to convert the homogeneous coordinates into inhomogeneous coordinates. Assume that the frame size in the original video is $w \times h$, and the coordinates of the top left corner and bottom right corner of the cropped rectangle in the stabilized video are $[c_1, d_1], [c_2, d_2]$, the hard constraint for video stabilization can be represented as

$$\begin{cases} 0 \leq \tilde{u}_{kj} \leq w \\ 0 \leq \tilde{v}_{kj} \leq h \end{cases}, \forall \begin{bmatrix} u_{kj} \\ v_{kj} \end{bmatrix} s.t. \begin{cases} c_1 \leq u_{kj} \leq c_2 \\ d_1 \leq v_{kj} \leq d_2 \end{cases} \tag{4.5}$$

No matter what real-time rotation smoothing method is used, the estimated smoothed rotation for each frame may violate the constraint (4.5). Therefore,

the original rotation estimate has to be projected onto the constraint set. The constraint (4.5) is very complex with respect to the rotation matrices that we want to compute and no algorithms as far as I know are guaranteed to handle it efficiently. I propose an simple approximate projection method:

$$\hat{\mathbf{R}} = \mathbb{P}(\hat{\mathbf{R}}^*) = \mathbf{R}\text{expm}(\beta^*\text{logm}(\mathbf{R}^{-1}\hat{\mathbf{R}}^*)), \tag{4.6}$$

where $\beta^* \in [0, 1]$ is the maximum possible value so that the projection result satisfies constraint (4.5). $\mathbf{R}$ is the original rotation matrix and $\hat{\mathbf{R}}^*$ is the initial estimate of the smoothed rotation matrix. In other words, we only search along the direction defined by the tangent vector $\text{logm}(\mathbf{R}^{-1}\hat{\mathbf{R}}^*)$. The proposed projection returns the exact solution if the constraint set is a geodesic ball around $\mathbf{R}$, which is a good approximation of the constraint set (4.5). In practice, β^* can be efficiently found by bisection search. During the search, whether the constraint is satisfied can be evaluated only using the four corner points of the cropped rectangle, because (4.3) is a homography transformation. The projected projection (4.6) can thus be implemented very fast. In the following sections I will use it to keep the online smoothing results satisfying the black-border constraint.

4.5 Rotation Smoothing via IIR Filtering

Assuming the 2D motion parameter for each frame k in the original video is $\boldsymbol{\theta}_k$, 1st-order IIR smoothing calculates the smoothed motion parameter $\hat{\boldsymbol{\theta}}_k$ by

$$\hat{\boldsymbol{\theta}}_k = \alpha\hat{\boldsymbol{\theta}}_{k-1} + (1-\alpha)\boldsymbol{\theta}_k, \tag{4.7}$$

where $\alpha \in [0, 1]$ is the smoothing coefficient. $\boldsymbol{\theta}_k$ is a vector in multi-dimensional Euclidean space. For instance, if 2D affine motion model is used, the dimension of $\boldsymbol{\theta}_k$ is six. If we use 3D rotational camera motion model, however, (4.7) is not appropriate because it is defined based on distance measure in Euclidean space instead of $\mathbf{SO(3)}$ manifold. In fact, the weighted sum on the right hand side of (4.7) does not necessarily return a valid 3D rotation matrix.

The 1st-order IIR filtering in (4.7), however, can be interpreted in another way

$$\hat{\boldsymbol{\theta}}_k = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \alpha||\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{k-1}||^2 + (1-\alpha)||\boldsymbol{\theta} - \boldsymbol{\theta}_k||^2. \tag{4.8}$$

In other words, $\hat{\boldsymbol{\theta}}_k$ is a linear interpolation between $\hat{\boldsymbol{\theta}}_{k-1}$ and $\boldsymbol{\theta}_k$ based on Euclidean distance. Given the geodesic distance defined on $\mathbf{SO(3)}$, we can naturally extend (4.8) as

$$\hat{\mathbf{R}}_k = \underset{\mathbf{R}}{\operatorname{argmin}}\, \alpha d_g(\mathbf{R}, \hat{\mathbf{R}}_{k-1})^2 + (1-\alpha) d_g(\mathbf{R}, \mathbf{R}_k)^2. \tag{4.9}$$

This just means a linear interpolation between $\hat{\mathbf{R}}_{k-1}$ and $\mathbf{R}_k$ on $\mathbf{SO(3)}$ based on the geodesic distance. It has been show that such interpolation is equivalent to spherical linear interpolation (slerp) on unit quaternion representation of 3D rotation matrices [77], which can be computed very fast. The interpolation result for every frame is projected by (4.6) before being used for the next frame. Algorithm 3 shows the proposed IIR-like 3D rotation smoothing. I use unit quaternions to represent the 3D rotations instead of matrices, but the representations can be easily converted from one to the other.

Algorithm 3 IIR-like 3D Rotation Smoothing

Input: $\mathbf{q}_1, \cdots, \mathbf{q}_K$ (original rotations)
Output: $\hat{\mathbf{q}}_1, \cdots, \hat{\mathbf{q}}_K$ (smoothed rotations)
$\hat{\mathbf{q}}_1 = \mathbf{q}_1$
for $k = 2$ **to** K **do**
    $\hat{\mathbf{q}}_k = \text{slerp}(\mathbf{q}_k, \hat{\mathbf{q}}_{k-1}, \alpha)$
    $\hat{\mathbf{q}}_k \leftarrow \mathbb{P}(\hat{\mathbf{q}}_k)$
end for

4.6 Rotation Smoothing via Unscented Kalman Filtering

A constant-velocity model defines the motion parameters and their velocities as state variables. The velocities are assumed to be constant in propagation except for a small random acceleration (usually modeled as Gaussian noise). The measurements are the original motion parameters and the smoothed parameters are just the estimated states. For 2D motions, a linear system is sufficient to model the dynamics and measurements so state estimate can be obtained precisely using Kalman filtering (assuming independent Gaussian process and measurement noise). Here I use the same idea to design a constant-velocity model for 3D rotation. Similarly, the state variable for each stage (frame) consists of the 3D rotation and the angular velocity. I still use unit quaternion representation of 3D rotations. The dynamic model is

$$\begin{bmatrix} \mathbf{q}_k \\ \boldsymbol{\omega}_k \end{bmatrix} = \begin{bmatrix} \mathbf{q}_{k-1} \otimes q(\boldsymbol{\omega}_{k-1}) \\ \boldsymbol{\omega}_{k-1} + \mathbf{w}_k \end{bmatrix}, \tag{4.10}$$

where $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ is the process noise corresponding to angular acceleration. $\otimes$ is the quaternion multiplication and the function $q(\boldsymbol{\omega}_{k-1})$ is used to convert $\boldsymbol{\omega}_{k-1}$ to a unit-quaternion-represented rotation. If the 3D rotation is

represented by matrices then $\mathbf{q}_{k-1} \otimes q(\boldsymbol{\omega}_{k-1})$ is equivalent to $\mathbf{R}_{k-1}\text{expm}(\boldsymbol{\omega}_{k-1})$. The measurement model is then

$$\tilde{\mathbf{q}}_k = \mathbf{q}_k \otimes q(\mathbf{v}_k), \tag{4.11}$$

where $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is the measurement noise and $\tilde{\mathbf{q}}_k$ is the original camera rotation for frame k. The system defined by (4.10) and (4.11) is actually defined based on geodesic distance on $\mathbf{SO}(3)$ and can be considered as a "linear" system on the manifold though linearity is not defined on manifolds. However, so far there is no efficient iterative algorithm to solve the estimation problem for such system exactly. In practice, we can solve the estimation problem on its embedded Euclidean space. In Euclidean space the system is clearly non-linear. In this chapter I use unscented Kalman filtering (UKF) to solve the problem.

UKF uses a deterministic sampling technique known as the unscented transform [42]. It picks a minimal set of sample points (sigma points) to represent the posterior probability of the state vector and propagates them through the non-linear dynamic and measurement functions, from which the mean and covariance of the estimate are then recovered. The unscented transform (UT) usually produces more accurate estimates than the analytic local linearization employed by the extended Kalman filter (EKF) for sequential nonlinear estimation with a small increase in computational complexity. Another choice is to use a particle filter [2]. A Particle filter is a general Monte Carlo (sampling) method for sequential estimation problem. It uses a large set of random sample

points to represent the posterior probability of the state vector. Compared to EKF and UKF, a particle filter can generate the most accurate estimates for nonlinear systems. However, the computational complexity of particle filter is much higher than EKF and UKF. Therefore, I use UKF to strike a good balance between estimation accuracy and computational complexity.

Algorithm 4 shows the general UKF steps for estimation of the nonlinear system

$$\begin{cases} \mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{v}_k) \\ \mathbf{y}_k = h(\mathbf{x}_k, \mathbf{n}_k) \end{cases}, \tag{4.12}$$

where $\mathbf{v}_k$ and $\mathbf{n}_k$ are zero-mean Gaussian noise with covariance $\mathbf{P}_v$ and $\mathbf{P}_n$.

In Algorithm 4, the original state vector is augmented with the process and measurement noise before sigma point generation. L is the dimension of the augmented state. λ is the composite scaling parameter and γ_i are unscented transform (UT) weights. These parameters are determined as

$$\begin{cases} \lambda = \alpha^2(L + \kappa) - L \\ \gamma_0^{(m)} = \lambda/(L + \lambda) \\ \gamma_0^{(c)} = \lambda/(L + \lambda) + (1 - \alpha^2 + \beta) \\ \gamma_i^{(m)} = \gamma_i^{(c)} = 1/[2(L + \lambda)], \quad i = 1, \ldots, 2L \end{cases} \tag{4.13}$$

In (4.13) α determines the spread of the sigma points around the mean and is usually set to a small positive value (e.g., 1e-3). κ is secondary scaling parameter which is usually set to 0, and β is used to incorporate prior knowledge of the distribution of the state vector (for Gaussian distributions, $\beta = 2$ is optimal). Details of UKF can be found in [42, 89].

Algorithm 4 Unscented Kalman Filter

Initialize with:

$\hat{\mathbf{x}}_0 = E[\mathbf{x}_0]$

$\mathbf{P}_0 = E[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^{\mathrm{T}}]$

$\hat{\mathbf{x}}_0^a = [\hat{\mathbf{x}}_0^{\mathrm{T}}\ \mathbf{0}\ \mathbf{0}]^{\mathrm{T}}$

$$\mathbf{P}_0^a = \begin{bmatrix} \mathbf{P}_0 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_v & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{P}_n \end{bmatrix}$$

for $k = 1$ **to** K **do**

Calculate sigma points:

$\mathcal{X}_{k-1}^a = [\hat{\mathbf{x}}_0^a\ \hat{\mathbf{x}}_0^a \pm \sqrt{(L+\lambda)\mathbf{P}_{k-1}^a}]$

Time update:

$\mathcal{X}_{k|k-1}^x = f(\chi_{k-1}^x, \mathcal{X}_{k-1}^v)$

$\hat{\mathbf{x}}_k^- = \sum_{i=0}^{2L} \gamma_i^{(m)} \mathcal{X}_{i,k|k-1}^x$

$\mathbf{P}_k^- = \sum_{i=0}^{2L} \gamma_i^{(c)} (\mathcal{X}_{i,k|k-1}^x - \hat{\mathbf{x}}_k^-)(\mathcal{X}_{i,k|k-1}^x - \hat{\mathbf{x}}_k^-)^{\mathrm{T}}$

$\mathcal{Y}_{k|k-1} = h(\mathcal{X}_{k|k-1}^x, \mathcal{X}_{k-1}^n)$

$\hat{\mathbf{y}}_k^- = \sum_{i=0}^{2L} \gamma_i^{(m)} \mathcal{Y}_{i,k|k-1}$

Measurement update equations:

$\mathbf{P}_{\mathbf{y}_k,\mathbf{y}_k} = \sum_{i=0}^{2L} \gamma_i^{(c)} (\mathcal{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-)(\mathcal{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-)^{\mathrm{T}}$

$\mathbf{P}_{\mathbf{x}_k,\mathbf{y}_k} = \sum_{i=0}^{2L} \gamma_i^{(c)} (\mathcal{X}_{i,k|k-1}^x - \hat{\mathbf{x}}_k^-)(\mathcal{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-)^{\mathrm{T}}$

$\mathcal{K} = \mathbf{P}_{\mathbf{x}_k,\mathbf{y}_k} \mathbf{P}_{\mathbf{y}_k,\mathbf{y}_k}^{-1}$

$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathcal{K}(\hat{\mathbf{y}}_k - \hat{\mathbf{y}}_k^-)$

$\mathbf{P}_k = \mathbf{P}_k^- - \mathcal{K}\mathbf{P}_{\mathbf{y}_k,\mathbf{y}_k}\mathcal{K}^{\mathrm{T}}$

end for

4.6.1 Estimate Projection

The quaternion estimate from UKF, however, does not necessarily have unit norm because the problem is solved in Euclidean space. Therefore, a normalization step is needed to make sure the mean estimate of 3D rotation has unit norm. In addition, the projection step 4.6 is still needed to guarantee that there is no black borders.

The 3D rotation $\mathbf{q}_k$ is not the only variable in the state vector. The angular velocity $\boldsymbol{\omega}_k$ is correlated with $\mathbf{q}_k$. For general constrained estimation problem, the state estimate is usually projected as

$$\mathbf{x}_k = \underset{\mathbf{x}}{\operatorname{argmin}}(\mathbf{x} - \mathbf{x}_k^*)^{\mathrm{T}}\mathbf{W}(\mathbf{x} - \mathbf{x}_k^*), \text{s.t. } \mathbf{x} \in \Phi, \tag{4.14}$$

where $\mathbf{x}_k^*$ is the unconstrained estimated mean, Φ is the constraint set. Matrix W is a positive-definite weighting matrix, which is usually chosen as $\mathbf{P}_k^{-1}$, the inverse of unconstrained covariance estimate. Such projection is able to obtain the maximum probability estimate of the state subject to the state constraints. In my case, only projecting the 3D rotation $\mathbf{q}_k$ using (4.6) and leaving $\boldsymbol{\omega}_k$ unchanged is equivalent to perform estimate projection with $\mathbf{W} = \mathbf{I}$ (identity matrix). It has been shown that projection with $\mathbf{W} = \mathbf{P}_k^{-1}$ can generate more accurate constrained state estimate. However, estimate projection with $\mathbf{W} = \mathbf{P}_k^{-1}$ is very difficult for our problem because of the complexity of the constraint set.

As a result, I propose an approximate estimate projection method. In the first step I still project the 3D rotation $\mathbf{q}_k$ using (4.6). Assume 3D rotation

before and after projection are $\hat{\mathbf{q}}_k^*$ and $\hat{\mathbf{q}}_k$, respectively. I then find the angular velocity estimate by

$$\hat{\boldsymbol{\omega}}_k = \underset{\boldsymbol{\omega}}{\operatorname{argmin}}[(\hat{\mathbf{q}}_k - \hat{\mathbf{q}}_k^*)^{\mathrm{T}}, (\boldsymbol{\omega} - \hat{\boldsymbol{\omega}}_k)^{\mathrm{T}}]\mathbf{P}_k^{-1} \begin{bmatrix} \hat{\mathbf{q}}_k - \hat{\mathbf{q}}_k^* \\ \boldsymbol{\omega} - \hat{\boldsymbol{\omega}}_k \end{bmatrix}, \tag{4.15}$$

where $\hat{\boldsymbol{\omega}}_k^*$ is the unconstrained angular velocity mean estimation and $\mathbf{P}_k$ is the unconstrained covariance estimate for the entire state vector. The result, although not optimal, has a much higher probability (measured by the objective function in (4.14)) compared to leaving angular velocity mean estimation unchanged. (4.15) is just a unconstrained quadratic programming problem and can be easily solved in closed form.

I summarize the proposed UKF-based rotation smoothing algorithm in Algorithm 5.

Algorithm 5 UKF-based 3D Rotation Smoothing

Input: $\mathbf{q}_1, \cdots, \mathbf{q}_K$ (original rotations)
Output: $\hat{\mathbf{q}}_1, \cdots, \hat{\mathbf{q}}_K$ (smoothed rotations)
Parameters: $\mathbf{Q}$, $\mathbf{R}$ (process and measurement noise variance)
for $k = 1$ **to** K **do**
    Obtain unconstrained UKF estimate $\hat{\mathbf{q}}_k^*$, $\hat{\boldsymbol{\omega}}_k^*$, $\mathbf{P}_k$
    $\hat{\mathbf{q}}_k^* = \hat{\mathbf{q}}_k^* / ||\hat{\mathbf{q}}_k^*||_2$ (normalization)
    $\hat{\mathbf{q}}_k \leftarrow \mathbb{P}(\hat{\mathbf{q}}_k^*)$
    Obtain $\hat{\boldsymbol{\omega}}_k$ by (4.15)
    (Mean and covariance estimate to pass to the next stage are $\hat{\mathbf{q}}_k$, $\hat{\boldsymbol{\omega}}_k$, $\mathbf{P}_k$)
end for

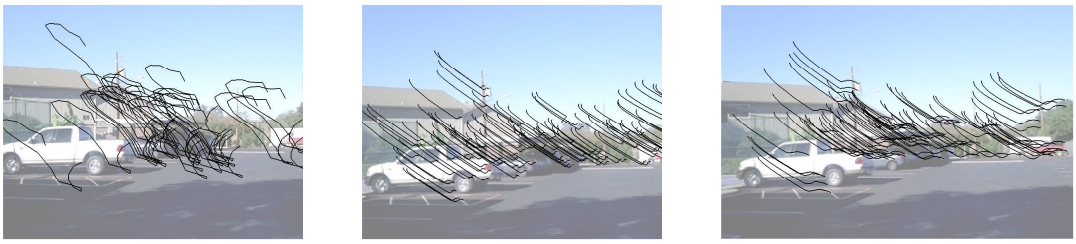
4.7 Experimental Results

Before showing the experimental results, I first discuss on how to choose the parameters for online motion smoothing. For IIR-like rotation smoothing, the only parameter to tune is the smoothing weight α in Algorithm 3. Clearly larger α tends to generate smoother rotation sequences without the black-border constraint. However, rotation smoothing with larger α tends to deviate farther from the original camera motion, and thus triggers estimate projection in Section 4.4 more frequently. The constraints are actually determined by the original (unsmooth) rotation and therefore differ across different frames. Frequent estimate projection may add the unwanted camera shake back and reduce the smoothness of the rotation smoothing output. In the following experiments, I fix the $\alpha = 0.95$ if not mentioned, which does not triggers estimate projection very often.

For the UKF-based rotation smoothing we need to choose the process and measurement noise covariance $\mathbf{Q}$ and $\mathbf{R}$. I smooth the test videos with the proposed offline smoothing method in Chapter 3 and use the results as the ground truth of the intentional (smooth) camera motion. The measurement noise covariance $\mathbf{R}$ can then be learned from the differences between the original and the smoothed rotation sequences. In all of our experiments I fix $\mathbf{R}$ as diag(0.002, 0.002, 0.002). The process noise covariance $\mathbf{Q}$ reflects the expected angular acceleration range and works similarly as α in the IIR-like rotation smoothing. I fix $\mathbf{Q}$ as diag(3e-10, 3e-10, 3e-10) to reach a balance between smoothness and less frequent estimate projection.

I show the video stabilization results with the proposed algorithms on two real videos. I refer the readers to the webpage of my related paper (`http://users.ece.utexas.edu/~bevans/papers/2014/stabilization/`) to view the original videos and all our results. Both videos are captured by a walking person using Google Nexus S smartphone on urban streets. The original frame size is 720×480. In our experiments, I use a 540×360 cropping size for the stabilized video. The camera rotation is obtained by integrating the gyroscope readings after sensor calibration and synchronization. Fig. 4.1 and 4.2 show a comparison between the original videos and the smoothed videos using the proposed algorithms. I use feature trajectories as a visualization of the changing frames. I detect Harris corner points in a certain frame and track them for 20 frames. The feature trajectories are plotted as black curves on top of the starting frame (the frames themselves are plotted using alpha channel 0.5 (more transparent) to make the curves clearer). Both proposed algorithms are able to effectively smooth the feature trajectories. Note that I detect and track the feature points independently in the videos so the location and number of the feature points can be different.

We compare original and smoothed camera rotation using the proposed IIR-like smoothing algorithm for video no. 2 in Fig. 4.3. In this figure, the camera rotation corresponding to each frame in the video sequence is shown in form of Tait-Bryan angles (a similar representation as Euler angles). Note that, however, the proposed motion smoothing is performed naturally on the manifold $\mathbf{SO}(\mathbf{3})$ instead of the Euclidean space of rotation angle representa-



(a) Original video (b) IIR-like smoothing (c) UKF-based smoothing

Figure 4.1: Stabilization comparison for video no. 1. Features are tracked from frame 31 to frame 50. The feature trajectories are plotted as black curves on frame 31.

tions.

To evaluate the the video stabilization algorithms numerically I compare the mean of L^1 norm of the angular velocity and acceleration of the 3D rotation sequences. L^1 norm is chosen because it is more robust to outliers compared to L^2 norm [29]. The numerical comparison is shown in Table. 4.1.

4.7.1 Comparison Against 2D motion Smoothing

I compare the proposed 3D rotational real-time motion smoothing algorithms with existing 2D motion smoothing algorithms. Compared to 2D models, the 3D rotational model can reflect the real camera motion more accurately and results in smoother results. Fig. 4.4 shows a comparison of feature trajectories between the proposed IIR-like rotation smoothing and IIR motion smoothing with 2D affine model (both using $\alpha = 0.95$). In the stabi-

(a) Original video (b) IIR-like smoothing (c) UKF-based smoothing

Figure 4.2: Stabilization comparison for video no. 2. Features are tracked from frame 46 to frame 65. The feature trajectories are plotted as black curves on frame 46.

lized videos using 2D affine model there are always wavy distortion because of the inaccuracy of the motion model. This distortion cannot be found in the stabilized videos using 3D rotation model.

4.7.2 Estimate Projection for UKF

In Section 4.6.1 I propose an approximate estimate projection method that also changes the estimate of the angular velocity $\hat{\boldsymbol{\omega}}_k$. In Table 4.2 I evaluate compare the proposed estimate projection in UKF-based rotation smoothing against the simple projection that does not change the estimate of the angular velocity. We can find that for both test videos the proposed estimate projection results in smoother rotation sequences.

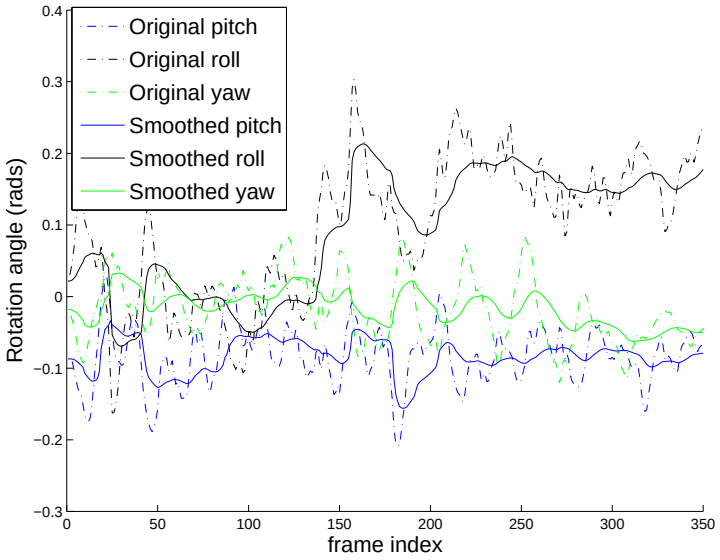


Figure 4.3: Comparison of the original and the smoothed camera rotation using the proposed IIR-like rotation smoothing algorithm for video no. 2.

4.7.3 IIR vs. UKF

The IIR-based and UKF-based methods have long been used to smooth 2D camera motion sequences (for 2D models the constant-velocity-based smoothing is solved via Kalman filtering instead of UKF because the system is linear). Both algorithms have been shown to effectively smooth camera motion in real time, whether for 2D motion models or 3D rotation model in this chapter. In terms of computational speed, the IIR-like smoothing algorithm is much faster. With MATLAB implementation on a 2.3GHz Intel i5 processor machine, the IIR-like smoothing algorithm takes only 1.54ms/frame while the UKF-based smoothing algorithm takes 6.97ms/frame.

Table 4.1: Numerical comparison between orignal videos and smoothed videos

Video no. 1		
	mean angular velocity	mean angular acceleration
Unsmoothed	0.0328	0.0270
IIR smoothing	0.00730	0.00408
UKF smoothing	0.00957	0.00413
Video no. 2		
	mean angular velocity	mean angular acceleration
Unsmoothed	0.0293	0.0256
IIR smoothing	0.00805	0.00331
UKF smoothing	0.00884	0.00303

If there are abrupt changes in the intentional camera motion, the UKF-based method tends to perform better with less frequent triggering of estimate projection. The reason is that in UKF-based smoothing algorithm we only assume that the angular velocity is almost constant, not that the angular velocity is almost zero. By estimating the angular velocity together with the smoothed rotation, we can better keep track of the change in intentional camera motion. This is shown in Table 4.3. In this table I compare the two algorithms for video no. 2, in which there is a sudden change (panning) of the intentional camera rotation. To make a fair comparison I tune the parameter α in the IIR-like smoothing algorithm from 0.95 to 0.9 so that it triggers the same times of estimate projection as the default UKF-based smoothing algorithm. From Table 4.3 we can find that under the same times of estimate projection the UKF-based smoothing algorithm generate smoother rotation sequence.

(a) 2D affine smoothing (b) 3D rotational smoothing

Figure 4.4: Stabilization comparison for video no. 2 between 2D affine smoothing and 3D rotational smoothing. Features are tracked from frame 246 to frame 265. The feature trajectories are plotted as black curves on frame 246.

4.8 Conclusions

In this chapter I propose two real-time motion smoothing algorithms for video stabilization using a pure 3D rotation motion model. Both algorithms directly smooth the 3D rotation sequences on the $\mathbf{SO(3)}$ manifold. The first algorithm is similar to 1st-order IIR filtering and requires only one slerp step for each frame. The second algorithm assumes a constant angular velocity model of the smooth rotation sequences and obtain the smooth rotation matrix for each frame via sequential probabilistic estimation. The estimation problem is solved efficiently using unscented Kalman filter. I also add a simple projection step to guarantee that no black borders intrude into the stabilized video frames. I have demonstrated in experiments that our algorithms are very fast and can generate better video stabilization results than their 2D counterparts.

Table 4.2: Numerical comparison between two estimate projection methods

Video no. 1		
	mean angular velocity	mean angular acceleration
simple projection	0.00102	0.00480
proposed projection	0.00957	0.00413
Video no. 2		
	mean angular velocity	mean angular acceleration
simple projection	0.00940	0.00431
proposed projection	0.00884	0.00303

Table 4.3: Numerical comparison between IIR-like and UKF-based rotation smoothing algorithms for video no. 2

	mean angular velocity	mean angular acceleration
IIR smoothing	0.00102	0.00367
UKF smoothing	0.00884	0.00303

Chapter 5

Conclusion

While more and more videos are shot by handheld cameras, the quality of these videos are severely affected by unwanted jitter and rolling shutter effect. In my dissertation I seek to rectify the rolling shutter effect and stabilize the videos and generate visually pleasant videos. I propose the following thesis statement:

For handheld cameras with CMOS sensors, videos can be satisfactorily rectified and then stabilized either online or offline, with the camera motion estimated directly from gyroscopes after effective sensor calibration.

In the following section, I discuss how my contributions in each chapter contribute toward defending this thesis statement.

5.1 Summary

Generally video stabilization and rolling shutter rectification require to (1) estimate the camera motion accurately, (2) compute new camera motion, and (3) synthesize new video frames. This dissertation focuses on the first two steps. An efficient online camera-gyroscope calibration algorithm is proposed to help obtain camera motion estimation directly from gyroscope.

Compared to vision-based motion estimation methods, gyroscope-aided methods can provide for more accurate camera motion estimation at higher time-resolution. For motion smoothing, I propose contributions to both offline and online motion smoothing methods. Offline motion smoothing is formulated as a regression problem on the manifold of 3D rotation matrices. Online motion smoothing is achieved via IIR-like filtering and also sequential probabilistic estimation with a constant-velocity model. For both offline and online motion estimation, I incorporate constraints to limit the appearance of black borders so that all of the pixels in the stabilized video frames are defined. These three contributions provide a complete solution of robust video stabilization and rolling shutter rectification for handheld cameras, which outperforms the state-of-the-art algorithms.

The specific contributions of this dissertation are concluded as below:

In Chapter 2 I propose an online calibration and synchronization algorithm for cellphones that is able to estimate not only the camera projection parameters, but also the gyroscope bias, the relative orientation between the camera and gyroscope, and the delay between the timestamps of the two sensors. The proposed algorithm is based on the generalization of the coplanarity constraint of the cross products of matched features in a rolling shutter camera model. The proposed algorithm can also be naturally extended to a global shutter camera model by forcing the readout time for each frame t_r to be zero. Monte Carlo simulation and experiments run on real data collected from cellphones show that the proposed algorithm can successfully estimate all of

the needed parameters with different kinds of motion of the cellphones. This online calibration and synchronization of rolling shutter camera and gyroscope make it more convenient for not only video stabilization, but also gyro-aided feature tracking, and visual-inertial navigation.

In Chapter 3 I propose a novel offline camera motion algorithm using a 3D rotational camera motion model. I exploit the manifold structure of not only the 3D rotation matrices, but also the sequences of 3D rotation matrices. This allows us to globally formulate motion smoothing as a regression problem based on geodesic distance. Furthermore, I force the solution to lie on a Cartesian product of geodesic balls so that every pixel in the stabilized frame is visible in the original frame. I directly solve the formulated problem on manifold by generalizing the existing two-metric projection algorithm in Euclidean space. The 3D camera rotation for each frame is obtained reliably using gyroscopes that are equipped in most smart phones and tablets, no matter whether there is motion blur or abrupt illumination change in the videos. I have demonstrated in experiments that our algorithm is very fast and can generate better video stabilization results than state-of-the-art methods.

In Chapter 4 I propose two real-time motion smoothing algorithms for video stabilization using a pure 3D rotation motion model with known camera projection parameters. Both proposed algorithms aim at smoothing 3D rotation matrix sequences in a causal way. The first algorithm smooths the 3D rotation sequences in a way similar to 1st-order IIR filtering. The second algorithm uses sequential probabilistic estimation under a constant angular

velocity model. These two algorithms are generalized from classical 2D motion smoothing algorithms. I exploit the manifold structure of the rotation matrices so that the proposed algorithms directly smooth the 3D rotation sequences on the manifold. In addition, I introduce a simple projection step in order to guarantee that no black borders intrude into the stabilized video frames. Experimental results show that the proposed algorithms are able to effectively stabilize video sequences and outperform their 2D counterparts with less jitter and distortion.

5.2 Future Work

In this section I outline several interesting research directions in video stabilization and rolling shutter rectification, potentially for other researchers interested in this area.

1. **Rolling shutter rectification for moving objects**:

   The rolling shutter rectification proposed in this dissertation can be understood as intra-frame video stabilization, which only works when the rolling shutter effect is caused by camera motion. However, object motion can also create rolling shutter effect even if the camera is still. For instance, a fast passing train may appear skewed while the background looks normal.

   General rolling shutter rectification was first proposed in [30]. Optical flows are estimated between adjacent frames and the rectified frame is

synthesized by linear interpolation of the two neighboring frames, where the interpolation weights are spatially variant (determined by the exposure time of each pixel). [85] uses a similar approach, but with a higher-order spline for frame interpolation. Such methods are able to rectify rolling shutter effects caused by either camera motion or object motion, but suffers from the extreme high computational complexity of optical flow.

It is well known that block-based motion estimation can be done in real time. In fact, block-based motion estimation has been successfully applied to solve a quite related problem : motion-compensated frame interpolation [9, 35, 44]. Motion-compensated frame interpolation aims at double the frame rate of the video for modern televisions. Compared to the optical-flow-based rolling shutter rectification, this frame interpolation uses a spatially-invariant interpolation weights of the two adjacent frames. I think there will be a natural extension from motion-compensated frame interpolation to rolling shutter rectification by varying the interpolation weights. Block-based motion estimation, although less accurate than optical flow, can dramatically accelerate rolling shutter rectification.

2. **Modeling of higher-order difference for 3D rotation sequences**:

   In my proposed offline motion smoothing algorithm, the objective function to minimize is the first-order difference of the 3D rotation sequences.

[29] has shown the advantage of adding higher-order difference into the objective function. Unlike first-order difference which can be defined on Riemannian manifolds by geodesic distance, higher-order difference are hard to generalize from Euclidean space to manifolds. [7] proposes to approximate the second-order difference on manifold as the Euclidean distance of first-order differences. Exact modeling of the higher-order difference on manifold with corresponding manifold optimization algorithms to minimize the objective is worth attempting.

3. **Quality evaluation of video stabilization and rolling shutter rectification**:

   Quality evaluation is very important for video stabilization and rolling shutter rectification. For rolling shutter rectification, we can compare the rectified frames with ground truth frames captured by global shutter cameras. However, obtaining ground truth is difficult since it requires accurate spatial calibration and temporal synchronization of different cameras. Synthetic simulation was applied in [75]. The videos were generated by synthetic rendering of the 3D scene created by 3D modeling software.

   Non-reference quality assessment for rolling shutter rectification is another way without knowing the ground truth. We can directly evaluate whether the frames are geometrically correct (whether all of the rows are captured at the same moment). For instance, we can fit a funda-

mental matrix using matching features between two different frames and compute the residual error after fitting.

Quality evaluation of video stabilization is even more difficult since it is related to human perception. People may have different standards in evaluating whether a video looks stable. Existing methods of video stabilization quality evaluation such as [68] were designed based on 2D translational camera motion model and may not be appropriate for recent video stabilization algorithms with 3D camera motion models. Moreover, to my knowledge, no automatic algorithms have been proposed for evaluating video stabilization quality with human perception taken into account. Many researchers compared their algorithms with others by conducting a user study and used the user preference as the evaluation metric. Usually the number of participants of the user studies was less than one hundred. Designing an automatic quality evaluation algorithm based on the results of user studies with more participants is desirable for not only convenience but also closer fitting to the opinions of the majority of viewers for a sufficient sample size.

4. **Black border constraint and viewing box**:

   In chapters 3 and 4, I use a viewing box to avoid black borders in the stabilized frames. In my dissertation, the size of the viewing box is fixed for all of the test video sequences. Video sequences differ a lot in amount of unwanted jitter. We can further change the viewing box size

for different videos. For example, if the original video sequence is already close to be stable, we can use a larger viewing box. This allows more content of the video to be kept. The size selection can be done either with human intervention, or automatically.

Very few studies have been published on automatic viewing box size selection. In [72] the authors proposed a dual pass motion smoothing method to try to find a cropping size as large as possible. An optimal size of the viewing box should consider both stability and the amount of content being kept. We may still need comprehensive user study and human visual perception knowledge to determine how to choose the optimal size. The aforementioned size selection is only possible for offline motion smoothing. For online motion smoothing, we still have to use a fixed size due to missing information of how camera motion changes afterward.

Once the viewing box size is selected, the proposed algorithms in Chapter 3 use it as a hard constraint on motion smoothing. If we can tolerate a few black borders showing in the viewing box, we may soften the constraints. The problem then becomes an unconstrained regression problem. The computational complexity of solving the problem is greatly reduced. For unconstrained regression, we have to choose the parameter α in (3.5) wisely instead of setting it to a very large number. Optimal selection of α, again, may rely on a comprehensive user study.

Appendix

Appendix 1

Derivation of Jacobian Matrices

In this appendix I derive how Jacobian matrices of the measurement equation can be computed analytically for Chapter 2. As shown in (2.19), the measurement equation $h()$ can be decomposed into several independent components $\{h_j()\}$ for each single coplanarity constraint. Therefore, we only need to show $\frac{\partial h_j}{\partial \mathbf{x}}$ and $\frac{\partial h_j}{\partial \mathbf{v}}$, where $\mathbf{v}$ contains both gyroscope measurement noise and feature detection noise.

Each single measurement equation $h_j()$ can be represented in form of (2.17). Let $\mathbf{a}_i$ denote $\mathbf{R}_i\mathbf{f}_i$ and let $\mathbf{b}_i$ denote $\mathbf{R}'_i\mathbf{f}'_i$. Then we have

$$h_j(\mathbf{x}, \mathbf{v}) = \det[(\mathbf{a}_1 \times \mathbf{b}_1)|(\mathbf{a}_2 \times \mathbf{b}_2)|(\mathbf{a}_3 \times \mathbf{b}_3)]. \tag{1.1}$$

$\frac{\partial h_j}{\partial \mathbf{x}}$ can be computed as $\sum_{i=1}^{3} \frac{\partial h_j}{\partial \mathbf{a}_i}\frac{\partial \mathbf{a}_i}{\partial \mathbf{x}} + \frac{\partial h_j}{\partial \mathbf{b}_i}\frac{\partial \mathbf{b}_i}{\partial \mathbf{x}}$. $\frac{\partial h_j}{\partial \mathbf{v}}$ can be computed in the same way. Without loss of generality, we only show how to compute $\frac{\partial h_j}{\partial \mathbf{b}_1}$, $\frac{\partial \mathbf{b}_1}{\partial \mathbf{x}}$ and $\frac{\partial \mathbf{b}_1}{\partial \mathbf{v}}$.

Based on the definition of matrix determinant we have

$$\frac{\partial h_j}{\partial \mathbf{b}_1} = [(\mathbf{a}_2 \times \mathbf{b}_2) \times (\mathbf{a}_3 \times \mathbf{b}_3)]^{\mathrm{T}}\mathrm{skew}(\mathbf{a}_1), \tag{1.2}$$

where skew() is defined as in (2.13).

To simplify the representation, we define

$$\mathbf{d}_1 = \begin{bmatrix} (1+\kappa_1 r^2+\kappa_2 r^4)(u'_{x_1}+v'_{x_1}-c_x) \\ (1+\kappa_1 r^2+\kappa_2 r^4)(u'_{y_1}+v'_{y_1}-c_y) \\ f \end{bmatrix} \tag{1.3}$$

and $\mathbf{e}_1 = \frac{1}{||\mathbf{d}_1||_2}\mathbf{d}_1$. Note that here

$$r = \sqrt{\left(\frac{u'_{x_1}+v'_{x_1}-c_x}{f}\right)^2 + \left(\frac{u'_{y_1}+v'_{y_1}-c_y}{f}\right)^2}. \tag{1.4}$$

In this way, we have

$$\mathbf{b}_1 = \mathbf{R}'_1(\mathbf{q}_c \otimes \mathbf{e}_1) \tag{1.5}$$

according to (2.17) and (2.18). The rotation matrix $\mathbf{R}'_1$ is not affected by the camera intrinsic parameters. So we have

$$\frac{\partial \mathbf{b}_1}{\partial c_x} = \mathbf{R}'_1[\mathbf{q}_c \otimes (\frac{\partial \mathbf{e}_1}{\partial \mathbf{d}_1}\frac{\partial \mathbf{b}_1}{\partial c_x})], \tag{1.6}$$

where

$$\frac{\partial \mathbf{b}_1}{\partial c_x} = \begin{bmatrix} -(2\kappa_1+4\kappa_2 r^2)\frac{(c_x-u'_{x_1}-v'_{x_1})^2}{f^2} - (1+\kappa_1 r^2+\kappa_2 r^4) \\ (2\kappa_1+4\kappa_2 r^2)\frac{(c_x-u'_{x_1}-v'_{x_1})(u'_{y_1}-v'_{y_1}-c_y)}{f^2} \\ 0 \end{bmatrix}. \tag{1.7}$$

Similarly we can obtain $\frac{\partial \mathbf{b}_1}{\partial c_y}$, $\frac{\partial \mathbf{b}_1}{\partial f}$, $\frac{\partial \mathbf{b}_1}{\partial \kappa_1}$ and $\frac{\partial \mathbf{b}_1}{\partial \kappa_2}$.

As mentioned in Section 2.5.4, we use a minimal 3-element error representation $\delta\boldsymbol{\theta}$ for $\mathbf{q}_c$ and have

$$\frac{\partial \mathbf{b}_1}{\partial \delta\boldsymbol{\theta}} = -\mathbf{R}'_1 \text{skew}(\mathbf{q}_c \otimes \mathbf{e}_1). \tag{1.8}$$

For more details about the minimal 3-element error representation please see [87].

Recall that the rotation matrix $\mathbf{R}'_1$ can be computed as in (2.10)

$$\mathbf{R}'_1 = \prod_{n=1}^{M} \Theta(\boldsymbol{\omega}_n \Delta t_n). \tag{1.9}$$

Different from (2.10), (1.9) only contains angular velocities with non-zero Δt_n. Similar to (2.11) which shows how to compute Δt_n for the example shown in Fig. 2.4, we have

$$\begin{cases} \Delta t_1 = \tau_2 - (T + t_d) \\ \Delta t_n = \tau_{n+1} - \tau_n, n = 2, \cdots, M-1 \\ \Delta t_M = (T + t_d + t_r \frac{u'_{y_1}}{h}) - \tau_M \end{cases} \tag{1.10}$$

where T is the framestamp for the frame in which the feature $[u'_{x_1}, u'_{y_1}]^{\mathrm{T}}$ appears. Please note that t_d and t_r only affect the value of Δt_1 and Δt_M.

By defining $\Gamma_n = \prod_{m=1}^{n-1} \Theta(\boldsymbol{\omega}_m \Delta t_m)$ and $\gamma_n = [\prod_{m=n+1}^{M} \Theta(\boldsymbol{\omega}_m \Delta t_m)](\mathbf{q}_c \otimes \mathbf{e}_1)$, we have

$$\mathbf{b}_1 = \Gamma_n \Theta(\boldsymbol{\omega}_n \Delta t_n) \gamma_n, \forall n = 1, \cdots, M. \tag{1.11}$$

It is not difficult to show that

$$\frac{\partial \mathbf{b}_1}{\partial \Delta t_n} = -\Gamma_n \mathrm{skew}(\gamma_n) \boldsymbol{\omega}_n. \tag{1.12}$$

Therefore, we can compute $\frac{\partial \mathbf{b}_1}{\partial t_d}$ and $\frac{\partial \mathbf{b}_1}{\partial t_r}$ as

$$\begin{cases} \frac{\partial \mathbf{b}_1}{\partial t_d} = -\Gamma_M \mathrm{skew}(\gamma_M) \boldsymbol{\omega}_M + \Gamma_1 \mathrm{skew}(\gamma_1) \boldsymbol{\omega}_1 \\ \frac{\partial \mathbf{b}_1}{\partial t_r} = -\frac{u'_{y_1}}{h} \Gamma_M \mathrm{skew}(\gamma_M) \boldsymbol{\omega}_M. \end{cases} \tag{1.13}$$

Given (2.14) we can compute $\frac{\partial \mathbf{b}_1}{\partial \mathbf{b}_g}$ as

$$\frac{\partial \mathbf{b}_1}{\partial \mathbf{b}_g} = \sum_{n=1}^{M} \frac{\partial \mathbf{b}_1}{\partial \boldsymbol{\omega}_n}, \tag{1.14}$$

where

$$\frac{\partial \mathbf{b}_1}{\partial \boldsymbol{\omega}_n} = -\Delta t_n \Gamma_n \text{skew}(\gamma_n). \tag{1.15}$$

So far we have derived the derivative $\frac{\partial \mathbf{b}_1}{\partial \mathbf{x}}$ analytically as in (1.6), (1.7), (1.8), (1.13) and (1.14). Next we compute the derivative of $\mathbf{b}_1$ with respect to the measurement noise.

The gyroscope measurement noise $\{\mathbf{n}_{gn}\}$ appears in (1.9) through (2.14). As a result we have

$$\frac{\partial \mathbf{b}_1}{\partial \mathbf{n}_{gn}} = \frac{\partial \mathbf{b}_1}{\partial \boldsymbol{\omega}_n} = -\Delta t_n \Gamma_n \text{skew}(\gamma_n). \tag{1.16}$$

The feature detection noise $\{\mathbf{v}_i\}$ appears in (1.3). Also note that $\mathbf{b}_1$ is only affected by v'_{x_1} and v'_{y_1}. As a result we have

$$\begin{cases} \frac{\partial \mathbf{b}_1}{\partial v'_{x_1}} = \mathbf{R}'_1[\mathbf{q}_c \otimes (\frac{\partial \mathbf{e}_1}{\partial \mathbf{d}_1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix})] \\ \\ \frac{\partial \mathbf{b}_1}{\partial v'_{y_1}} = \mathbf{R}'_1[\mathbf{q}_c \otimes (\frac{\partial \mathbf{e}_1}{\partial \mathbf{d}_1} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix})] \end{cases} . \tag{1.17}$$

In this way, the derivative $\frac{\partial \mathbf{b}_1}{\partial \mathbf{v}}$ can be computed analytically.

Bibliography

[1] P.-A. Absil, R. Mahony, and R. Sepulchrer. *Optimization Algorithm on Matrix Manifolds.* 2008.

[2] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filter for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Trans. Signal Processing*, 50(2):174–188, 2002.

[3] T. Bailey and H. Durrant-Whyte. Simultaneous localisation and mapping (SLAM): Part II state of the art. *IEEE Robotics & Automation Magazine*, 13:108–117, 2006.

[4] S. Baker, E. Bennett, Kang S. B., and R. Szeliski. Removing rolling shutter wobble. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, June 2010.

[5] D.P. Bertsekas. Projected Newton methods for optimization problems with simple constraints. *SIAM Journal Control and Opt.*, 20(2):221–246, March 1982.

[6] D.P. Bertsekas. *Nonlinear Programming: 2nd ed.* 1999.

[7] N. Boumal and P.-A. Absil. A discrete regression method on manifolds and its application to data on so(n). In *Proc. IFAC World Congress*, 2011.

[8] C. Buehler, M. Bosse, and L. McMillan. Non-metric image-based rendering for video stabilization. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 609–614, December 2001.

[9] B-D. Choi, J-W. Han, C-S. Kim, and S-J. Ko. Motion-compensated frame interpolation using bilateral motion estimation and adaptive overlapped block motion compensation. *IEEE Trans. Circuits and Systems for Video Technology*, 17:407–416, 2007.

[10] J. Civera, D.R. Bueno, A.J. Davison, and J.M.M. Montiel. Camera self-calibration for sequetial Bayesian structure from motion. In *Proc. IEEE Intl. Conf. Robotics and Automation*, 2009.

[11] P. Corke, J. Lobo, and J. Dias. An introduction to inertial and visual sensing. *Intl. Journal Robotics Research*, 26(6):519–535, 2007.

[12] A.J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proc. IEEE Intl. Conf. Computer Vision*, volume 2, pages 1403–1410, October 2003.

[13] J.C. Dunn. A subspace decomposition principle for scaled gradient projection methods: global theory. *SIAM Journal Control and Opt.*, 29(5):1160–1175, 1991.

[14] J.C. Dunn. A subspace decomposition principle for scaled gradient projection methods: local theory. *SIAM Journal Control and Opt.*, 31(1):219–246, 1993.

[15] H. Durrant-Whyte and T. Bailey. Simultaneous localisation and mapping (SLAM): Part I the essential algorithms. *IEEE Robotics & Automation Magazine*, 13:99–110, 2006.

[16] A. Edelman, T.A. Arias, and S.T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal Matrix Anal. Appl.*, 20(2):303–353, 1998.

[17] S. Ertürk. Image sequence stabilization: motion vector integration (MVI) versus frame position smoothing (FPS). In *Proc. Intl. Symp. Image and Signal Processing and Analysis*, June 2001.

[18] S. Ertürk. Real-time digital image stabilization using Kalman filters. *Real-Time Imaging*, 8:317–328, 2002.

[19] S. Ertürk and T.J. Dennis. Image sequence stabilization based on DFT filtering. *IEE Proc. Vision, Image and Signal Processing*, 147(2):95–102, 2000.

[20] R. Ferreira, J. Xavier, J.P. Costeira, and V. Barroso. Newton method for Riemannian centroid computation in naturally reductive homogeneous spaces. In *Proc. IEEE Intl. Conf. Acoustics, Speech and Signal Processing*, May 2006.

[21] M.A. Fischler and R.C. Bolles. Random sample consensus, a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[22] P.T. Fletcher, C. Lu, and S. Joshi. Statistics of shape via principal geodesic analysis on Lie groups. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, June 2003.

[23] P.-E. Forssén and E. Ringaby. Rectifying rolling shutter video from hand-held devices. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, June 2010.

[24] E.M. Gafni and D.P. Bertsekas. Two-metric projection methods for constrained optimization. *SIAM Journal Contral and Opt.*, 22(6):936–964, 1984.

[25] J. Gallier. *Notes on Differential Geometry and Lie Groups*. University of Pennsylvannia, 2012.

[26] C. Geyer, M. Meingast, and S. Sastry. Geometric models of rolling-shutter cameras. In *Proc. Workshop Omnidirectional Vision*, 2005.

[27] V. Govindu. Lie-algebraic avearaging for globally consistent motion estimation. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, June 2004.

[28] M. Grundmann, V. Kwatra, D. Castro, and I. Essa. Calibration-free rolling shutter removal. In *Proc. IEEE Intl. Conf. Computational Photography*, April 2012.

[29] M. Grundmann, V. Kwatra, and Ifran Essa. Auto-directed video stabilization with robust L1 optimal camera paths. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, June 2011.

[30] J. Gu, Y. Hitomi, T. Mitsunaga, and S.K. Nayar. Coded rolling shutter photography: Flexible space-time sampling. In *Proc. IEEE Intl. Conf. Computational Photography*, March 2010.

[31] G. Hanning, N. Forslöw, P.-E. Forssén, E. Ringaby, D. Törnqvist, and J. Callmer. Stabilizing cell phone video using inertial measurement sensors. In *Proc. IEEE Intl. Workshop Mobile Vision*, November 2011.

[32] R.I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.

[33] P.S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics & Applications*, 6(11):56–67, 1986.

[34] Y.-F. Hsu, C.-C. Chou, and M.-Y. Shih. Moving camera video stabilization using homography consistency. In *Proc. IEEE Intl. Conf. Image Processing*, September 2012.

[35] A-M. Huang and T.Q. Nguyen. A multistage motion vector processing method for motion-compensated frame interpolation. *IEEE Trans. Image Processing*, 17:694–708, 2008.

[36] M. Hwangbo, J.-S. Kim, and T. Kanade. Gyro-aided feature tracking for a moving camera: fusion, auto-calibration and GPU implementation. *Intl. Journal Robotics Research*, 30(14):1755–1774, 2011.

[37] C. Jia and B. L. Evans. Video demonstrations for constrained 3D rotation smoothing via global manifold regression for video stabilization. `http://users.ece.utexas.edu/~bevans/papers/2015/stabilization/`.

[38] C. Jia and B. L. Evans. Probabilistic 3-D motion estimation for rolling shutter video rectification from visual and inertial measurements. In *Proc. IEEE Intl. Workshop Multimedia Signal Processing*, September 2012.

[39] C. Jia and B. L. Evans. 3D rotational video stabilization using manifold optimization. In *Proc. IEEE Intl. Conf. Acoustics, Speech, and Signal Processing*, May 2013.

[40] C. Jia and B. L. Evans. Online calibration and synchronization of cellphone camera and gyroscope. In *Proc. IEEE Global Conf. Signal and Information Processing*, December 2013.

[41] C. Jia and B. L. Evans. Constrained 3D rotation smoothing via global manifold regression for video stabilization. *IEEE Trans. Signal Processing*, Accepted for publication with mandatory minor revisions.

[42] S.J. Julier and J.K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92:401–422, 2004.

[43] S.-H. Jung and C.J. Taylor. Camera trajectory estimation using inertial sensor measurements and structure from motion results. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, volume 2, pages 732–737, December 2001.

[44] S-J. Kang, K-R. Cho, and Y.H. Kim. Motion compensated frame rate up-conversion using extended bilateral motion estimation. *IEEE Trans. Consumer Electronics*, 53:1759–1767, 2007.

[45] H. Karcher. Riemannian center of mass and mollifier smoothing. *Comm. Pure Appl. Math*, 30:509–541, 1977.

[46] A. Karpenko, D. Jacobs, J. Baek, and M. Levoy. Digital video stabilization and rolling shutter correction using gyroscopes. Technical report, Stanford University, March 2011.

[47] J. Kelly and G.S. Sukhatme. Visual-inertial sensor fusion: localization, mapping and sensor-to-sensor self-calibration. *Intl. Journal Robotics Research*, 30(1):56–79, 2011.

[48] L. Kneip, A. Martinelli, S. Weiss, D. Scaramuzza, and R. Siegwart. Closed-form solution for absolute scale velocity determination combining inertial measurements and a single feature correspondence. In *Proc. IEEE Intl. Conf. Robotics and Automation*, May 2011.

[49] L. Kneip, R. Siegwart, and M. Pollefeys. Finding the exact rotation between two images independently of the translation. In *Proc. European*

Conf. Computer Vision, October 2012.

[50] J. Lee and S.Y. Shin. General construction of time-domain filters for orientation data. *IEEE Trans. Visualization and Computer Graphics*, 8:119–128, 2002.

[51] K.-Y. Lee, Y.-Y. Chuang, Chen B.-Y., and M. Ouhyoung. Video stabilization using robust feature trajectories. In *Proc. IEEE Intl. Conf. Computer Vision*, pages 1397–1404, September 2009.

[52] M Li, B. Kim, and A.I. Mourikis. 3-D motion estimation and online temporal calibration for camera-IMU systems. In *Proc. IEEE Intl. Robotics and Automation*, May 2013.

[53] M Li, B. Kim, and A.I. Mourikis. Real-time motion tracking on a cellphone using inertial sensing and a rolling shutter camera. In *Proc. IEEE Intl. Robotics and Automation*, May 2013.

[54] C-K. Liang, L-W. Chang, and H. Chen. Analysis and compensation of rolling shutter effect. *IEEE Trans. Image Processing*, 17(8):1323–1330, August 2008.

[55] A. Litvin, J. Konrad, and W. Karl. Probabilistic video stabilization using Kalman filtering and mosaicking. *Proc. IS&T/SPIE Symp. Electronic Imaging, Image and Video Comm. and Proc.*, pages 663–674, 2003.

[56] F. Liu, M. Gleicher, H. Jin, and A. Agarwala. Content-preserving warps for 3D video stabilization. *ACM Trans. Graphics*, 28(3), 2009.

[57] F. Liu, M. Gleicher, J. Wang, H. Jin, and A. Agarwala. Subspace video stabilization. *ACM Trans. Graphics*, 30(1), 2011.

[58] J. Lobo and J. Dias. Relative pose calibration between visual and inertial sensors. *Intl. Journal Robotics Research*, 26(6):561–575, 2007.

[59] S. Lovegrove, A. Patron-Perez, and G. Sibley. A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras. In *Proc. British Machine Vision Conf.*, September 2013.

[60] B. Lucas and T. Kanade. An iterative image registration technique with application to stereo vision. In *Proc. Intl. Joint Conf. Artificial Intelligence*, pages 674–679, 1981.

[61] A. Martinelli. Vision and IMU data fusion: closed-form solutions for attitude, speed, absolute scale, and bias determination. *IEEE Trans. Robotics*, 28(1):44–60, 2012.

[62] Y. Matsushita, E. Ofek, W. Ge, X. Tang, and H.-Y. Shum. Full-frame video stabilization with motion inpainting. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28, July 2006.

[63] F.M. Mirzaei and S.I. Roumeliotis. A Kalman filter-based algorithm for IMU-camera calibration. *IEEE Trans. Robotics*, 24(5):1143–1156, 2008.

[64] J. More. The Levenberg-Marquardt algorithm, implementation and theory. *Numerical Analysis*, pages 105–116, 1977.

[65] C. Morimoto and R. Chellappa. Fast 3D stabilization and mosaic construction. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, June 1997.

[66] A. I. Mourikis and S. I. Roumeliotis. On the treatment of relative-pose measurements for mobile robot localization. In *Proc. IEEE Intl. Conf. Robotics and Automation*, May 2006.

[67] A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint Kalman filter for vision-aided inertial navigation. In *Proc. IEEE Intl. Conf. Robotics and Automation*, April 2007.

[68] M. Niskanen, O. Silven, and M. Tico. Video stabilization performance assessment. In *Proc. IEEE Intl. Conf. Multimedia and Expo*, July 2006.

[69] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In *Proc. IEEE Intl. Conf. Computer Vision and Pattern Recognition*, April 2004.

[70] J. Nocedal and S.J. Wright. *Numerical Optimization*. 1999.

[71] L. Oth, P. Furgale, L. Kneip, and R. Siegwart. Rolling shutter camera calibration. In *Proc. IEEE Intl. Conf. Computer Vision and Pattern Recognition*, June 2013.

[72] P. Pan, A. Minagawa, J. Sun, Y. Hotta, and S. Naoi. A dual pass video stabilization system using iterative motion estimation and adaptive motion smoothing. In *Proc. Intl. Conf. Pattern Recognition*, August 2010.

[73] M. Pilu. Video stabilization as a variational problem and numerical solution with the Viterbi method. In *Proc. IEEE Intl. Conf. Computer Vision and Pattern Recognition*, June 2004.

[74] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch. Visual modeling with a hand-held camera. *Intl. Journal Computer Vision*, 59(3):207–232, 2004.

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

[76] J. Shi and C. Tomasi. Good features to track. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 593–600, June 1994.

[77] K. Shoemake. Animating rotation with quaternion curves. In *Proc. ACM SIGGRAPH*, pages 245–254, 1985.

[78] D. Simon. Kalman filtering with state constraints: a survey of linear and nonlinear algorithms. *IET Control Theory and Applications*, 4:1303–1318, 2010.

[79] D. Simon and D.L. Simon. Aircraft turbofan engine health estimation using constrained Kalman filtering. *ASME Journal of Engineering for Gas Turbines and Power*, 127:323–328, 2005.

[80] B.M. Smith, L. Zhang, H. Jin, and A. Agarwala. Light field video stabilization. In *Proc. IEEE Intl. Conf. Computer Vision*, September 2009.

[81] S.T. Smith. *Geometric optimization methods for adaptive filtering*. PhD thesis, Harvard University, 1993.

[82] C. Song, H. Zhao, W. Jing, and Y. Bi. Robust video stabilization based on bounded path planning. In *Proc. Intl. Conf. Pattern Recognition*, November 2012.

[83] D. Strelow and S. Singh. Online motion estimation from image and inerital measurements. In *Proc. Workshop Integration of Vision and Inertial Sensors*, June 2003.

[84] D. Strelow and S. Singh. Motion estimation from image and inertial measurements. *Intl. Journal Robotics Research*, 23(12):1157–1195, 2004.

[85] Y. Sun and G. Liu. Rolling shutter distortion removal based on curve interpolation. *IEEE Trans. Consumer Electronics*, 58:1045–1050, 2012.

[86] M. Tico and M. Vehvilainen. Constraint motion filtering for video stabilization. In *Proc. IEEE Intl. Conf. Image Processing*, September 2005.

[87] N. Trawny and S.I. Roumeliotis. Indirect Kalman filter for 3D attitude estimation. *Univ. of Minnesota Tech. Report*, 2005.

[88] C. Udriste. *Convex Functions and Optimization Methods on Riemannian Manifolds*. 1994.

[89] E.A. Wan and R. Van der Merwe. The unscented Kalman filter for nonlinear estimation. In *Proc. Adaptive Systems for Signal Processing, Communications, and Control Symposium*, October 2000.

[90] Y-S. Wang, F. Liu, P-S. Hsu, and T-Y. Lee. Spatially and temporally optimized video stabilization. *IEEE Trans. Visualization and Computer Graphics*, 19(8):1354–1361, 2013.

[91] K. Yamaguchi. mexopencv. `http://www.cs.stonybrook.edu/~kyamagu/mexopencv/`.

[92] J. Yang, D. Schonfeld, and M. Mohamed. Robust video stabilization based on particle filter tracking of projected camera motion. *IEEE Trans. Circuits and Systems for Video Technology*, 19(7):945–954, 2009.

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

Vita

Chao Jia received the B.S. degree in Electrical Engineering from Tsinghua University, China in 2009 and the M.S. degree in Electrical and Computer Engineering from the University of Texas at Austin in 2011. Since then, he has been pursuing his PhD. at the same university. In summers of 2011, 2012 and 2013, he was an intern in Qualcomm Research in San Diego, California. At Qualcomm he was involved in low-light photography for cellphone cameras and cooperative RAN for real-time video streaming. His research interests include image and video processing, computational photography and machine learning applications in computer vision. He is a member of IEEE.

Permanent address: 3500 Greystone Dr. #139
Austin, Texas 78731

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.