Examensarbete

Video Stabilization and Rolling Shutter Correction using Inertial Measurement Sensors

Examensarbete utfört i Datorseende
vid Tekniska högskolan vid Linköpings universitet
av

Gustav Hanning

LiTH-ISY-EX--11/4464--SE
Linköping 2011
Video Stabilization and Rolling Shutter Correction using Inertial Measurement Sensors

Examensarbete utfört i Datorseende vid Tekniska högskolan i Linköping
av

Gustav Hanning

LiTH-ISY-EX--11/4464--SE

Handledare: **Erik Ringaby**
isy, Linköpings universitet

Examinator: **Per-Erik Forssén**
isy, Linköpings universitet
Linköping, 7 June, 2011
Gyrobaserad videostabilisering och korrektion för rullande slutare

Video Stabilization and Rolling Shutter Correction using Inertial Measurement Sensors

Most mobile video-recording devices of today, e.g. cell phones and music players, make use of a rolling shutter camera. A rolling shutter camera captures video by recording every frame line-by-line from top to bottom of the image, leading to image distortions in situations where either the device or the target is moving. Recording video by hand also leads to visible frame-to-frame jitter.

In this thesis, methods to decrease distortion caused by the motion of a video-recording device with a rolling shutter camera are presented. The methods are based on estimating the orientation of the camera from gyroscope and accelerometer measurements.

The algorithms are implemented on the iPod Touch 4, and the resulting videos are compared to those of competing stabilization software, both commercial and free, in a series of blind experiments. The results from this user study shows that the methods presented in the thesis perform equal to or better than the others.

Keywords: Rolling Shutter, Video Stabilization, Inertial Measurement Sensor
Abstract

Most mobile video-recording devices of today, e.g. cell phones and music players, make use of a rolling shutter camera. A rolling shutter camera captures video by recording every frame line-by-line from top to bottom of the image, leading to image distortions in situations where either the device or the target is moving. Recording video by hand also leads to visible frame-to-frame jitter.

In this thesis, methods to decrease distortion caused by the motion of a video-recording device with a rolling shutter camera are presented. The methods are based on estimating the orientation of the camera from gyroscope and accelerometer measurements.

The algorithms are implemented on the iPod Touch 4, and the resulting videos are compared to those of competing stabilization software, both commercial and free, in a series of blind experiments. The results from this user study shows that the methods presented in the thesis perform equal to or better than the others.

Sammanfattning

Mobiltelefoner, mp3-spelare och andra bärbara enheter som kan spela in film har ofta en kamera med rullande slutare. En sådan kamera fångar varje bild rad för rad, från topp till botten. Detta resulterar i distortion i bilden om antingen enheten eller objekt i bilden rör sig. Att filma för hand introducerar också skakighet i filmen.

I det här examensarbetet presenteras metoder för att minska den distortion som uppstår till följd av att en enhet med rullande slutare förflyttas under inspelning av en film. Metoderna bygger på estimering av kamerans orientering, utgående från mätdata från gyroskop och accelerometer.

Algoritmerna har implementerats på en iPod Touch 4 och de resulterande filmerna har jämförts med de från konkurrerande program i en serie blindtester. Resultaten från denna undersökning visar att metoderna som presenteras i examensarbetet är lika bra eller bättre än de övriga.
Acknowledgments

I would like to thank my examiner Per-Erik Forssén and my supervisor Erik Ringaby for their help and guidance during the work on this thesis. Thanks also to Nicklas Forslöw, with whom I have had many fruitful discussions.
Contents

1 Introduction 1
  1.1 Background ............................................. 1
  1.2 Problem Formulation .................................. 2
  1.3 Purpose and Goal ...................................... 2
  1.4 Limitations ............................................ 3
  1.5 Related Work .......................................... 3
  1.6 Report Outline ........................................ 4

2 Theory 5
  2.1 Camera Model .......................................... 5
    2.1.1 The Pinhole Camera Model ....................... 5
    2.1.2 The Projective Plane \( P^2 \) ................... 6
    2.1.3 The Camera Calibration Matrix .................. 7
  2.2 Motion and Sensor Models .............................. 7
    2.2.1 Time Continuous Motion Model ................. 8
    2.2.2 Measurement Noise ................................ 8
    2.2.3 Time Discrete Motion Model .................... 9
    2.2.4 Sensor Model ..................................... 9
    2.2.5 Estimation ....................................... 10
  2.3 Rectification and Stabilization ....................... 10
    2.3.1 More on Quaternions ............................. 11
    2.3.2 Image Rectification .............................. 12
    2.3.3 Video Stabilization .............................. 13
  2.4 Synchronization ....................................... 14

3 System Overview 19
  3.1 Hardware .............................................. 19
  3.2 Objective-C .......................................... 20
    3.2.1 Syntax .......................................... 20
    3.2.2 Memory Management .............................. 21
  3.3 OpenGL ES ............................................ 21
    3.3.1 Shaders .......................................... 22
    3.3.2 An OpenGL ES Example ......................... 22
4 Implementation
  4.1 Functionality ................................................. 25
  4.2 Recording .................................................. 28
  4.3 Filtering ...................................................... 28
  4.4 Rectification and Stabilization ............................... 29
    4.4.1 OpenGL ES .............................................. 29

5 Evaluation ....................................................... 33
  5.1 User Study .................................................... 33
    5.1.1 Deshaker .................................................. 36
    5.1.2 iMovie ‘11 ................................................. 36
    5.1.3 Movie Stiller .............................................. 36
    5.1.4 The iPhone/iPod Application ............................... 37
    5.1.5 Method .................................................... 37
    5.1.6 Ranking ................................................... 37
    5.1.7 Results .................................................... 38
  5.2 Discussion .................................................... 42
  5.3 Future Work .................................................. 43

Bibliography ......................................................... 45

A User Study Form .................................................. 47
Chapter 1

Introduction

In this chapter the background of the thesis is given together with a description of the problems it aims to solve. The purpose and goal are formulated and the limitations of the thesis are discussed. The chapter ends with a short summary of related research and an outline of the rest of the report.

1.1 Background

Most mobile video-recording devices of today, e.g. cell phones and music players, make use of a rolling shutter (RS) camera. A RS camera captures video by recording every frame line-by-line from top to bottom of the image. This is in contrast to a global shutter where an entire frame is recorded at once.

The RS technique gives rise to image distortions in situations where either the device or the target is moving. Figure 1.1 shows an example of how an image is distorted when using a rolling shutter. Here, vertical lines such as the flag poles appear slanted as a result of moving the camera quickly from left to right during recording. Recording video by hand also leads to visible frame-to-frame jitter. The recorded video is perceived as “shaky” and is not very enjoyable to watch.

Since mobile video-recording devices are so common, there is an interest in correcting these types of distortions. The sensors present in some of these devices, such as accelerometers and gyroscopes, provide a new way of doing this. Using sensor data it is possible to estimate the position and/or orientation of the device during recording and then use this information to do appropriate post-processing of a recorded video.
1.2 Problem Formulation

The process of correcting distortions caused by motion of the device using sensor data can be divided into three main steps:

1. Simultaneously collect video, audio and sensor data.
2. Estimate position and/or orientation of the device from collected sensor data.
3. Use the estimates to compensate for device motion.

In the first step, data from the sensors is gathered while recording a video on the device. In the case where the available sensors are accelerometer and gyroscope, the gathered data is acceleration and angular velocity measurements. The next step is obtaining the best possible estimates of the quantities of interest. This can be done by deriving mathematical models for the sensors and the motion of the device and then use these in a filter to get the estimates.

The third and last step is using the filter estimates to perform image processing on each frame of the video, in a way that removes as much as possible of the distortions described in section 1.1. This thesis will focus on steps 1 and 3 above, leaving the estimation step to a parallel thesis by Nicklas Forslöw [1].

1.3 Purpose and Goal

The purpose of the thesis is to implement and evaluate a video stabilization algorithm and a rolling shutter correction algorithm on a mobile video-recording device.
using gyroscope and accelerometer data to track the orientation of the device. The device used in this thesis is an iPod Touch 4, which is similar to the iPhone 4 in terms of hardware.

The goal is to create an iPhone/iPod application that decreases the image distortion caused by the motion of the device and its rolling shutter camera. The application should also be able to stabilize the recorded video. The generated sequences should match in quality those from both commercial products such as iMovie ’11 [2] and free applications such as Deshaker [3].

1.4 Limitations

This thesis will focus on correcting RS distortion generated by motion of the device rather than motion of the target. The sensor data does not provide any information about how possible targets are moving. Additional image processing would have to be performed to acquire this information.

The algorithms that rectify and stabilize a video sequence will not be applied in real time on the iPhone/iPod. High resolution video recording capabilities together with inferior processing power (compared to a personal computer) make this difficult. Also, the thesis will focus exclusively on the iPod Touch 4 and the iPhone 4. The algorithms should however be applicable on most video-recording devices with an accelerometer and a gyroscope.

As mentioned in section 1.2, the problem of obtaining good estimates of the state of the device from sensor data will not be covered in detail in this thesis. A more thorough explanation of this issue is available in [1].

1.5 Related Work

Forssén and Ringaby [4] present a method for rectifying video sequences from RS cameras. They model the distortions as being caused by the 3D motion of the camera and use a KLT tracker to estimate this motion. This method will be used as a starting point for the work on rolling shutter correction in this thesis. The theory behind the method is presented in more detail in chapter 2.

Joshi et al. [5] present a model to recover the camera rotation and translation by integrating the measured accelerations and angular velocities of a camera platform using accelerometers and gyroscopes. This information is then used to remove motion blur from images.
1.6 Report Outline

**Chapter 2** presents the theory behind the methods for orientation estimation, video stabilization and image rectification.

**Chapter 3** gives an overview of the iPod Touch 4 and the iPhone 4. The programming language Objective-C and the graphics API OpenGL ES are also presented.

**Chapter 4** describes the implementation of the algorithms from chapter 2 on the iPod Touch 4 and the iPhone 4.

**Chapter 5** presents a user study comparing the iPhone/iPod application to other stabilization software. It also contains a discussion on the results of the thesis and ideas for future work on the topic.
Chapter 2

Theory

This chapter presents the theory behind the most important concepts of the thesis: camera orientation estimation, image rectification and video stabilization. The chapter starts with the derivation of a camera model, inspired by [6].

In the next section, motion and sensor models are given together with an algorithm (the extended Kalman filter) to estimate the orientation of a device from sensor data. Much of the information in this section is taken from [7]. A more detailed description of how the estimation is done in practice can be found in [1].

The chapter ends with an overview of the methods used for image rectification and video stabilization, which are based on the work in [4], followed by a section on how to synchronize a recorded video with the collected sensor data.

2.1 Camera Model

A camera can be regarded a mapping from the 3D world to a 2D image. If the map is linear it can be represented by a matrix, which is often desirable. There are multiple ways to model the camera. A commonly used model is the pinhole camera model, where the camera aperture is described as a point.

2.1.1 The Pinhole Camera Model

Consider the setup in figure 2.1. The camera aperture is the origin $O$ of camera coordinate system $XYZ$. There is an image plane $Z = -f$, where $f$ is the focal length of the pinhole camera. The image plane has a 2D coordinate system $xy$. A point $P = (p_1, p_2, p_3)^T$ in space is projected onto the point $p$, where the ray from $P$ through $O$ intersects the image plane. The ray is described by $\lambda P$, $\lambda \in \mathbb{R}$, and at the intersection we have $\lambda p_3 = -f$, so $\lambda = -f/p_3$. 
Figure 2.1. The pinhole camera model.

$P$ is thus projected onto $p = (-fp_1/p_3, -fp_2/p_3)^T$ (image coordinates). This map from $\mathbb{R}^3$ to $\mathbb{R}^2$ is not linear, and hence it cannot be represented by a matrix. To get a linear map, we must think of image points as elements of $\mathbb{P}^2$, the 2D projective plane.

2.1.2 The Projective Plane $\mathbb{P}^2$

The projective plane $\mathbb{P}^2$ can be viewed as the set of all lines through the origin of $\mathbb{R}^3$. An element $\mathbf{x} \in \mathbb{P}^2$ then has three components, $\mathbf{x} = (x, y, z)^T$, describing the direction of the line. Two elements $\mathbf{x}$ and $\mathbf{y}$ are defined to be equal if $\mathbf{y} = \lambda \mathbf{x}$, where $\lambda$ is a non-zero real number. One can associate each line $\mathbf{x} \in \mathbb{P}^2$ having $z \neq 0$ with a point in $\mathbb{R}^2$ by

$$
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix} \mapsto \begin{pmatrix} x/z \\ y/z \end{pmatrix}, \tag{2.1}
$$

i.e. the first two components of the intersection point between the line and the plane $z = 1$. Lines with $z = 0$ are parallel to this plane and correspond to points at infinity. Conversely, one can go from $\mathbb{R}^2$ to $\mathbb{P}^2$ by

$$
\begin{pmatrix}
x \\
y
\end{pmatrix} \mapsto \lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \lambda \neq 0. \tag{2.2}
$$
2.2 Motion and Sensor Models

2.1.3 The Camera Calibration Matrix

If we again consider the situation in figure 2.1, we can now associate each point in the image plane with a line in $P^2$ as above. The projection can be seen as a linear map from $\mathbb{R}^3$ to $P^2$, given by

$$
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix} =
\begin{pmatrix}
  -f & 0 & 0 \\
  0 & -f & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  p_1 \\
  p_2 \\
  p_3
\end{pmatrix}.
$$

(2.3)

The line belonging to the point $p$ in the image plane is thus $(-fp_1, -fp_2, p_3)^T$, called the homogenous coordinates of $p$. The matrix expression in equation 2.3 assumes that the origin of the $xy$ coordinate system is centered at the intersection between the image plane and the $Z$ axis. If this is not the case, $P$ is projected onto $p = (-fp_1/p_3 - p_x, -fp_2/p_3 - p_y)^T$, where $p_x$ and $p_y$ are the translations of the $x$ and $y$ axis respectively. Equation 2.3 can now be rewritten

$$
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix} =
\begin{pmatrix}
  -f & 0 & -p_x \\
  0 & -f & -p_y \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  p_1 \\
  p_2 \\
  p_3
\end{pmatrix} = K
\begin{pmatrix}
  p_1 \\
  p_2 \\
  p_3
\end{pmatrix}.
$$

(2.4)

The matrix $K$ is called the camera calibration matrix. If we also invert the $x$ and $y$ axis, so that the image gets the right orientation, and introduce scale factors $m_x$ and $m_y$ that relate pixels to distance, we instead get

$$
K =
\begin{pmatrix}
  m_x f & 0 & m_x p_x \\
  0 & m_y f & m_y p_y \\
  0 & 0 & 1
\end{pmatrix} =
\begin{pmatrix}
  \alpha_x & 0 & x_0 \\
  0 & \alpha_y & y_0 \\
  0 & 0 & 1
\end{pmatrix},
$$

(2.5)

where $\alpha_x$ and $\alpha_y$ are the focal lengths of the camera in the $x$ and $y$ directions, given in pixels. $(x_0, y_0)^T$ is the focal point, also measured in pixels. The $K$ matrix is different for each type of camera, and also varies with the resolution chosen on a device. As an example, the camera calibration matrix for the iPod Touch 4 using a resolution of 1280x720 pixels is

$$
K \approx
\begin{pmatrix}
  1223.0 & 0 & 650.4 \\
  0 & 1217.5 & 345.3 \\
  0 & 0 & 1
\end{pmatrix}.
$$

(2.6)

2.2 Motion and Sensor Models

In this section, a method for estimating the orientation of a video-capturing device from gyroscope and accelerometer data is presented. A more detailed description is given in [1].
2.2.1 Time Continuous Motion Model

The orientation of the device can be represented by a unit quaternion. A unit quaternion \( q \) is a four-dimensional vector, \( q = (q_0, q_1, q_2, q_3)^T \), with \( ||q|| = 1 \).

A rotation of angle \( \alpha \) about a three-dimensional unit vector \( u \) corresponds to quaternion \( q = (\cos(\alpha/2), \sin(\alpha/2)u) \). \( \text{(2.7)} \)

Now let \( q \) be a function of time, \( q = q(t) \), representing the orientation of the device relative to a fixed coordinate system. Then, one can derive an expression (see [7]) of the derivative of \( q(t) \) with respect to \( t \):

\[
\dot{q}(t) = \frac{1}{2} \begin{pmatrix}
0 & -\omega_x & -\omega_y & -\omega_z \\
\omega_x & 0 & \omega_z & -\omega_y \\
\omega_y & -\omega_z & 0 & \omega_x \\
\omega_z & \omega_y & -\omega_x & 0 \\
\end{pmatrix} q(t) = \frac{1}{2} S_1(\omega) q(t). \quad \text{(2.8)}
\]

This is the time continuous dynamic model or motion model of the device. \( \omega = (\omega_x, \omega_y, \omega_z)^T \) denotes the angular velocity of the device relative to a fixed system, expressed in the device’s own coordinate system. In this thesis, the angular velocity is assumed to be time-varying, \( \omega = \omega(t) \), and measured by a gyroscope.

2.2.2 Measurement Noise

The gyroscope does not measure \( \omega(t) \) exactly, but instead introduces measurement noise \( v(t) \), which will affect the motion model. Here, the noise is modeled as normally-distributed with zero mean, \( v(t) \sim N(0, Q) \). Now, note that equation 2.8 can be written as

\[
\dot{q}(t) = \frac{1}{2} \begin{pmatrix}
-q_1(t) & -q_2(t) & -q_3(t) \\
q_0(t) & -q_3(t) & q_2(t) \\
q_1(t) & q_0(t) & -q_1(t) \\
-q_2(t) & q_1(t) & q_0(t) \\
\end{pmatrix} \omega(t) = \frac{1}{2} S_2[q(t)] v(t). \quad \text{(2.9)}
\]

Then, with the gyroscope measuring \( \omega(t) + v(t) \), we get

\[
\dot{q}(t) = \frac{1}{2} S_1[\omega(t) + v(t)] q(t) = \frac{1}{2} S_1[\omega(t)] q(t) + \frac{1}{2} S_2[q(t)] v(t). \quad \text{(2.10)}
\]
2.2.3 Time Discrete Motion Model

The time continuous motion model in equation 2.10 must be discretized if one wants to use it in a time discrete filter implementation. Assuming that the angular velocity is constant during sampling intervals of length $T$, the following discrete model can be obtained [7]:

$$
q_t = \left( \cos \left( \frac{||\omega_t - 1||T}{2} \right) I + \frac{\sin \left( \frac{||\omega_t - 1||T}{2} \right)}{||\omega_t - 1||} S_1(\omega_t - 1) \right) q_{t-1} + \\
\sin \left( \frac{||\omega_t - 1||T}{2} \right) S_2(q_{t-1})v_{t-1}.
$$

(2.11)

For high sampling rates ($||\omega_t - 1||T$ small) this can be simplified to

$$
q_t = \left( I + \frac{T}{2} S_1(\omega_t - 1) \right) q_{t-1} + \frac{T}{2} S_2(q_{t-1})v_{t-1}.
$$

(2.12)

2.2.4 Sensor Model

Since the gyroscope measurements are already incorporated in the motion model, the only sensor left to model is the accelerometer. The accelerometer measures the acceleration along the axes of the device’s coordinate system. Both the Earth’s gravitational field and the acceleration caused by the user moving the device will affect these measurements. The measurement equation can be written

$$
y_t = a^d_t - g^d + e_t = R(q_t)(a^f_t - g^f) + e_t = h(q_t, a^f_t) + e_t,
$$

(2.13)

where $y_t = (y_1, y_2, y_3)^T$ is the measured acceleration along the three axes of the device. $a^d_t$ and $g^d$ are the user-induced acceleration and the gravity vector respectively, expressed in device coordinates, whereas $a^f_t$ and $g^f$ are expressed in a fixed coordinate system. $e_t$ is measurement noise and $R(q_t)$ is the rotation matrix relating the two coordinate systems. The rotation matrix can be computed from the quaternion $q_t$ [8] by

$$
R(q_t) =
\begin{pmatrix}
q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\
2(q_1q_2 - q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 + q_0q_1) \\
2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2
\end{pmatrix}.
$$

(2.14)
\( h(q_t, a_t') \) is a nonlinear function in \( q_t \). Since nothing is known about the user acceleration \( a_t' \) it is set to 0. Then, letting \( g = (0, 0, -g)^T \) and performing a first order Taylor expansion, we get

\[
y_t = h(q_t) + e_t \approx h(\hat{q}_{t|t-1}) + H_t (q_t - \hat{q}_{t|t-1}) + e_t,
\]

where \( \hat{q}_{t|t-1} \) is the predicted value of \( q_t \) given all measurements up to time \( t - 1 \) (as explained in section 2.2.5) and

\[
H_t = \left. \frac{\partial h(q_t)}{\partial q_t} \right|_{q_t = \hat{q}_{t|t-1}} = 2g \begin{pmatrix} -\dot{\hat{q}}_{2,t|t-1} & \dot{\hat{q}}_{3,t|t-1} & -\dot{\hat{q}}_{0,t|t-1} & \dot{\hat{q}}_{1,t|t-1} \\ \dot{\hat{q}}_{1,t|t-1} & \dot{\hat{q}}_{0,t|t-1} & \dot{\hat{q}}_{3,t|t-1} & \dot{\hat{q}}_{2,t|t-1} \\ \dot{\hat{q}}_{0,t|t-1} & -\dot{\hat{q}}_{1,t|t-1} & -\dot{\hat{q}}_{2,t|t-1} & \dot{\hat{q}}_{3,t|t-1} \end{pmatrix}.
\]

A linear measurement equation can now be acquired as

\[
\tilde{y}_t = y_t - h(\hat{q}_{t|t-1}) + H_t \hat{q}_{t|t-1} \approx H_t q_t + e_t.
\]

### 2.2.5 Estimation

An *extended Kalman filter* (EKF) can be used to estimate the state of the device from sensor data. If we write the motion model in equation 2.12 as

\[
q_t = F_{t-1} q_{t-1} + G_{t-1} v_{t-1},
\]

and use the sensor models in equations 2.13 and 2.17 we get the EKF algorithm 1 given below. The noise terms \( v_{t-1} \) and \( e_t \) are assumed to be zero-mean normally-distributed with covariance matrices \( Q_{t-1} \) and \( R_t \), respectively. The initial orientation of the device, \( q_0 \), is also assumed to be normally-distributed, \( q_0 \sim \mathcal{N}(q_i, P_i) \).

### 2.3 Rectification and Stabilization

In this section, the methods used for rectifying and stabilizing a video sequence are presented. It is assumed that we have managed to get “good enough” estimates of the orientation of the camera at certain time instances, as described in the previous section. Since the orientation estimates are given as unit quaternions, some more theory on these is needed.
2.3 Rectification and Stabilization

Algorithm 1: Extended Kalman filter
1. Initialize: set $\hat{q}_{0|0} = q_i$ and $P_{0|0} = P_i$.
2. Time update:
\[
\hat{q}_{t|t-1} = F_{t-1} \hat{q}_{t-1|t-1} \\
P_{t|t-1} = F_{t-1}P_{t-1|t-1}F_{t-1}^T + G_{t-1}Q_{t-1}G_{t-1}^T \\
\hat{q}_{t|t-1} = \hat{q}_{t|t-1}/||\hat{q}_{t|t-1}||
\]
3. Measurement update:
\[
K_t = P_{t|t-1}H_t^T(H_tP_{t|t-1}H_t^T + R_t)^{-1} \\
\hat{q}_{t|t} = \hat{q}_{t|t-1} + K_t(y_t - h(\hat{q}_{t|t-1})) \\
P_{t|t} = (I - K_tH_t)P_{t|t-1} \\
\hat{q}_{t|t} = \hat{q}_{t|t}/||\hat{q}_{t|t}||
\]
4. Set $t := t + 1$ and repeat from step 2.

2.3.1 More on Quaternions

Multiplication of two quaternions $p$ and $q$ is defined by
\[
p \odot q = \begin{pmatrix} p_0 \\ p_3 \end{pmatrix} \odot \begin{pmatrix} q_0 \\ q_3 \end{pmatrix} = \begin{pmatrix} p_0q_0 - p_3 \cdot q_3 \\ p_0q_3 + q_0p_3 + p_3 \times q_3 \end{pmatrix},
\]
where $p_3$ and $q_3$ are the last three components of $p$ and $q$. The inverse of $q$ is a quaternion $q^{-1}$ such that
\[
q^{-1} \odot q = q \odot q^{-1} = \begin{pmatrix} 1 \\ 0 \end{pmatrix},
\]
and can be computed by
\[
q^{-1} = \begin{pmatrix} q_0 \\ -q_3 \end{pmatrix}/||q||.
\]
If $q$ is a unit quaternion, it represents a rotation. The result of rotating a three-dimensional vector $v$ about $q$’s rotation axis (see equation 2.7) is
\[
\begin{pmatrix} 0 \\ v \end{pmatrix} = q^{-1} \odot \begin{pmatrix} 0 \\ v \end{pmatrix} \odot q.
\]
From this expression, it follows that rotating by $q_1$ and then by $q_2$ is the same as rotating by $q_1 \odot q_2$.
(q_1 \circ q_2)^{-1} \circ \begin{pmatrix} 0 \\ v \end{pmatrix} \circ (q_1 \circ q_2) = q_2^{-1} \circ \begin{pmatrix} 0 \\ v \end{pmatrix} \circ q_1 \circ q_2. \quad (2.23)

Interpolation between two unit quaternions can be performed in a number of ways, the simplest being standard linear interpolation (LERP):

\[
\text{LERP}(q_1, q_2, w) = (1 - w)q_1 + wq_2, \quad w \in [0, 1]. \quad (2.24)
\]

The linear interpolation must be followed by a normalization of the interpolated quaternion, to make sure that we stay on the four-dimensional unit sphere. A method that does not have this problem is SLERP, or spherical linear interpolation:

\[
\text{SLERP}(q_1, q_2, w) = \frac{\sin[(1 - w)\Omega]}{\sin \Omega} q_1 + \frac{\sin[w\Omega]}{\sin \Omega} q_2, \quad w \in [0, 1], \quad (2.25)
\]

where \( \Omega \) is the angle between \( q_1 \) and \( q_2 \), calculated as \( \Omega = \arccos(q_1 \cdot q_2) \). Another advantage of spherical linear interpolation is that the interpolated unit quaternion moves with constant speed along the arc from \( q_1 \) to \( q_2 \) as \( w \) changes uniformly from 0 to 1, allowing for a more smooth transition. More information about SLERP can be found in [8].

### 2.3.2 Image Rectification

Using homogenous coordinates as described in section 2.1.3, the relation between a point in space \( X \) (given in camera coordinates) and its projection onto the image plane \( x = (x, y, z)^T \) is

\[
x = KX. \quad (2.26)
\]

Since the columns of the camera calibration matrix are linearly independent, it is invertible and we can write

\[
X = \lambda K^{-1}x, \quad (2.27)
\]

where \( \lambda \) is a non-zero real number. If an image is captured with a rolling shutter camera, the rows are read at different time instances and at each of these instances the camera has a certain orientation. We want to transform the image to make it look like all rows were captured at once. Now assume we have estimated the orientation of the camera at the time instances when the first and last rows of an image were read. Let these estimates be unit quaternions \( q_f \) and \( q_l \), respectively. The change in orientation from first to last row is
2.3 Rectification and Stabilization

\[ q_{fl} = q_{f}^{-1} \odot q_{f}. \]  \hspace{1cm} (2.28)

The rotation from the first row to an arbitrary row can be interpolated from \( q_{fl} \) by using either linear or, preferably, spherical linear interpolation, with interpolation parameter \( w = y/(zN) \), \( N \) being the total number of rows in the image.

\[ q(x) = (S)\text{LERP}(q_{1}, q_{fl}, \frac{y}{zN}), \text{ where } q_{1} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \]  \hspace{1cm} (2.29)

We can rotate \( X \) back to its position during the acquisition of the first row of the frame, by the quaternion multiplication

\[ \begin{pmatrix} 0 \\ X' \end{pmatrix} = q(x)^{-1} \odot \begin{pmatrix} 0 \\ X \end{pmatrix} \odot q(x), \]  \hspace{1cm} (2.30)

and it follows that the new position in the image will be

\[ x' = KX'. \]  \hspace{1cm} (2.31)

2.3.3 Video Stabilization

Using the rectification technique described above, all rows are aligned to the top row. If one wants another reference row, it is possible to multiply with the quaternion corresponding to this row. For example, alignment to the middle row is performed by

\[ \begin{pmatrix} 0 \\ X'' \end{pmatrix} = q_{fm} \odot \begin{pmatrix} 0 \\ X' \end{pmatrix} \odot q_{fm}^{-1}, \]  \hspace{1cm} (2.32)

provided that an estimate \( q_{fm} \) of the camera orientation for this row is available. As before, the new location in the image is computed as \( x'' = KX'' \).

The reference orientation of a frame can be thought of as the orientation that we wish the camera would have had during capture of the frame. It can be changed without interfering with the rectification of the image.

If a video is recorded by hand, it will be shaky and the reference row orientations will vary a lot from frame to frame. To remove these quick changes in orientation, i.e. to stabilize the video, one could low-pass filter the reference quaternions.

Let \( q_{1}, \ldots, q_{m} \) be the unit quaternions corresponding to the reference row of each frame of a video consisting of \( m \) frames. A Hanning window of width \( N \),
can be applied component-wise to the quaternions by

\[
q'_n = \sum_{k=-\infty}^{\infty} w_{n-k} q_k, \quad n = 1, \ldots, m, \tag{2.34}
\]

followed by re-normalization, \(q''_n = q'_n / ||q'_n||\). A suitable window width is \(N \approx 100\), but this will vary from video to video. The resulting sequence \(q'_1, \ldots, q'_m\) can now be used as in equation 2.32, after multiplication by \(q_f^{-1}\). This will give a video with less frame-to-frame jitter and more smooth rotations. The theory behind this averaging of unit quaternions can be found in [9].

One problem with this approach is that the output video will not follow any quick changes in orientation present in the original video, leading to large areas in frames where the pixel values are unknown. This problem can be avoided by using a variable filter size, where a small filter is applied when there are fast rotations, and a larger filter is applied otherwise. The cumulative sum (CUSUM) algorithm can be used to detect the quick changes in orientation. The CUSUM algorithm is described in [10], and information on how it is used in the context of video stabilization can be found in [1].

Figure 2.2 shows the first component of the reference row quaternions for a video, before and after low-pass filtering using adaptive filter size. Here, the small filter has \(N = 29\) and the larger one has \(N = 99\). When swapping filters, the filter size is increased or decreased gradually to get smooth transitions. As seen, the jitter is suppressed and at the same time the low-pass filtered curve follows the original when there is a quick rotation at \(t > 810\).

### 2.4 Synchronization

Synchronizing the sensor data to the recorded video is important as failure to do so leads to a poor result, often worse than the original video. Both sensor data and frames are time stamped on the iPod Touch 4, but there is an unknown delay between the two. The time instance associated with a point \(x = (x, y, z)^T\) in a frame is

\[
t(x) = t_f + t_d + t_r \frac{y}{zN}, \tag{2.35}
\]

where \(t_f\) is the frame time stamp, \(t_d\) is the unknown delay and \(t_r\) is the readout time of the camera, which is assumed to be known. \(N\) is the number of rows in the
2.4 Synchronization

Figure 2.2. The first component, $q_0$, of reference row quaternions for a sample video.

The readout time, $t_r$, can be acquired by recording a flashing light source with a known frequency [4].

To find $t_d$, a number of features are identified in each of $M$ consecutive frames of the video. This can be done using the Shi-Tomasi corner detector [11] implemented in OpenCV [12]. Figure 2.3 shows 20 features identified in the first frame of a video sequence.

The features found in a frame are tracked to the next frame by the use of a KLT tracker [13], also available in OpenCV. They are then re-tracked to the original frame and only those that return to their original position, within a threshold, are kept. For each resulting point correspondence $x \leftrightarrow y$, we rotate $y = (u, v, w)^T$ back to the position it had at time instance $t(x)$:

$$ Y = \lambda K^{-1} y $$  \hspace{1cm} (2.36)

$$ \begin{pmatrix} 0 \\ Y' \end{pmatrix} = q[t(x)] \odot q[t(y)]^{-1} \odot \begin{pmatrix} 0 \\ Y \end{pmatrix} \odot q[t(y)] \odot q[t(x)]^{-1} $$  \hspace{1cm} (2.37)

$$ y' = K Y' $$  \hspace{1cm} (2.38)
Figure 2.3. Features identified by OpenCV method GoodFeaturesToTrack.

$q[t(x)]$ and $q[t(y)]$ are the orientations of the camera at time instances $t(x)$ and $t(y)$, acquired by interpolation from the EKF estimates. A cost function $J$ can now be defined as

$$J = \sum_k d(x_k, y'_k)^2,$$

where $d(x, y)$ is the pixel distance between $x$ and $y$,

$$d(x, y)^2 = \left( \frac{x}{z} - \frac{u}{w} \right)^2 + \left( \frac{y}{z} - \frac{v}{w} \right)^2.$$

Since $J$ could possibly have local minima, one can use a grid search to find the global minimum. Figure 2.4 shows $J$ as a function of $t_d$, for a sample video. As seen, there is a distinct minimum at $t_d = 0.144$ and two other, local, minima.
Figure 2.4. The cost function $J$, evaluated at different values of $t_d$. 
Chapter 3

System Overview

This chapter starts with an overview of the hardware used when implementing the rectification and stabilization methods from chapter 2. It also contains a presentation of the Objective-C programming language and the OpenGL ES graphics API, with some basic examples.

3.1 Hardware

The devices used in this thesis are the iPod Touch 4 and the iPhone 4. The two devices are similar in terms of hardware, and their most important specifications are given in table 3.1. One improvement from previous models, and important for this thesis, was the introduction of a three-axis gyroscope. The gyroscope measures the rotation rates around the axes as shown in figure 3.1. The maximum update frequency is approximately 60 Hz.

Both the iPod Touch 4 and the iPhone 4 also have an accelerometer, that measures the acceleration of the device along each of the axes in figure 3.1. The maximum update rate of the accelerometer is almost 100 Hz. The iPhone 4 has an additional sensor that the iPod Touch 4 lacks, namely a magnetometer, that measures the strength of magnetic fields.

<table>
<thead>
<tr>
<th>Processor</th>
<th>ARM Cortex-A8 CPU at 1 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>256 MB (iPod) or 512 MB (iPhone) DRAM</td>
</tr>
<tr>
<td>Graphics</td>
<td>PowerVR SGX535 GPU</td>
</tr>
<tr>
<td>Camera (main)</td>
<td>Supports 720p video recording at 30 fps</td>
</tr>
<tr>
<td>Display</td>
<td>3.5 inch, 960x640 resolution</td>
</tr>
</tbody>
</table>

Table 3.1. Hardware specifications for the iPod Touch 4 and the iPhone 4
3.2 Objective-C

The programming language used to create applications for the iPhone and iPod is Objective-C. It is a set of extensions to the standard C language, designed to give C full object-oriented programming capabilities [14]. Since Objective-C is based on C, all existing C libraries and tools can be used without modification [15].

3.2.1 Syntax

The syntax of Objective-C is somewhat different from other C-like languages. Instead of calling a method, one sends a message to an object. In code, this would be

```c
[anObject someMethod:argument];
```

The interface and implementation of a class, defined in header files (.h) and method files (.m), are declared as

```c
@interface someClass : superClass
{
    (variableType)instanceVariable;
}
+(returnType)classMethod;
-(returnType)instanceMethodWithParameter:(parameterType)parameter;
@end
```
3.3 OpenGL ES

and

```objective-c
@implementation someClass
+ (returnType)classMethod {
    // implementation
}
− (returnType) instanceMethodWithParameter :( parameterType )
    parameter {
        // implementation
    }
@end
```

The plus and minus signs indicate whether the method is a class method or an instance method. A class method can be called without an instance of that class, i.e.

```
[someClass classMethod];
```

while an instance method requires an instance of the class:

```
[classInstance instanceMethodWithParameter:parameter];
```

### 3.2.2 Memory Management

Although Objective-C offers support for automatic memory management in the form of a garbage collector, it is not available on the iPhone’s operating system iOS. Instead, a scheme known as reference counting is used. When an object is created it is assigned a retain count, starting at 1. By using the retain and release methods one can increase or decrease the retain count. When the retain count reaches zero, the object is destroyed.

```
// class instance has retain count 1 after creation.
someClass *classInstance = [[someClass alloc] init];
// Use the instance to do some work.
[classInstance instanceMethodWithParameter:parameter];
// Use release when done with the object. The retain count becomes 0 and the object is destroyed.
[classInstance release];
```

### 3.3 OpenGL ES

`OpenGL ES` (Open Graphics Library for Embedded Systems) is a graphics API for embedded systems such as mobile phones and video game consoles. It is used for visualizing 2D and 3D data and is a subset of standard desktop OpenGL,
optimized for handheld devices [16]. The iPod Touch 4 and the iPhone 4 support both versions 1.1 and 2.0 of OpenGL ES.

3.3.1 Shaders

OpenGL ES 2.0 introduced the ability to use shaders. Shaders are special computer programs running directly on the graphics hardware and are written in the OpenGL ES Shading Language (GLSL ES). There are two types of shaders: vertex shaders and fragment shaders. A vertex is a corner point of a triangle or some other polygon. A vertex shader operates on each such vertex in a scene, calculating the screen coordinate at which it will appear.

A fragment shader (also known as pixel shader) computes the color of each fragment (pixel) of the output image, but cannot change its position. The fragment shader is executed in succession to the vertex shader.

Variables in GLSL ES are defined to be in one of four different categories:

- **const** A compile-time constant.
- **attribute** Per-vertex data, for example vertex position.
- **uniform** Data that does not change across the primitive being processed, for example a projection matrix relating scene coordinates to screen coordinates.
- **varying** Data passed on from the vertex shader to the fragment shader, for example texture position.

Primitives are the simplest types of geometric objects that OpenGL ES can handle. They are points, lines and triangles.

3.3.2 An OpenGL ES Example

Figure 3.2 shows the output of a simple OpenGL ES example application. A triangle with vertices in \((-0.75, -0.75)^T, (0.75, -0.75)^T\) and \((-0.75, 0.75)^T\) was created and the vertices were colored red, blue and green. In this example, the vertex shader simply sets the position of the vertex and passes the chosen color on to the fragment shader. As seen, the color of pixels inside the triangle is interpolated from the colors of the vertices.
3.3 OpenGL ES

Figure 3.2. OpenGL ES example output.

Vertex shader

```
attribute vec4 a_position;
attribute vec4 a_color;

varying vec4 v_color;

void main()
{
    gl_Position = a_position;
    v_color = a_color;
}
```

Fragment shader

```
varying vec4 v_color;

void main()
{
    gl_FragColor = v_color;
}
```
Chapter 4

Implementation

An application able to record a video and log sensor data simultaneously and then rectify and stabilize the video have been created. The application runs on the iPod Touch 4 and the iPhone 4. In this chapter, the most important implementation details are presented.

The first section describes the basic functionality of the application, with screen captures to show the different views. In the following sections, a brief overview is given on how the recording of a video, the filtering of sensor data and the rectification and stabilization are performed.

4.1 Functionality

The application has two main uses; recording videos, and then stabilizing and rectifying them. Figure 4.1 shows the interface of the application when recording a video. Touching the screen starts and stops the recording. When done capturing, the user can find the recorded video in the “Sessions” tab, as depicted in figure 4.2.

Choosing a session in this tab shows details about the session, such as when the original video was recorded and which quality was used. The “Session Details” screen also lists the original video together with rectified and stabilized versions of it, see figure 4.3. From this screen, one can choose to stabilize and rectify the original video by pushing the “Stabilize Video” button.

The stabilization screen, shown in figure 4.4, lets the user set the amount of stabilization prior to starting the rectification and stabilization process. Internally, the slider value determines the size of the window function, described in section 2.3.3. Pushing the “Start” button initiates the process and a progress bar shows how much of the video has been rectified and stabilized, see figure 4.5.
Figure 4.1. The record screen of the application.

Figure 4.2. The sessions screen of the application.
4.1 Functionality

Figure 4.3. The session details screen of the application.

Figure 4.4. The stabilization screen of the application.
4.2 Recording

Videos can be captured in one of three different resolutions or quality settings: low (480x360), medium (640x480) and high (1280x720). When using the “high” setting, the camera captures 24 frames per second. In lower resolutions, 30 frames per second are recorded. At the same time, accelerometer and gyroscope data is logged, at a frequency of 60 Hz.

When capturing a new video, a folder is created in the device’s file system. After the recording is done, the sensor data is written to text files and a thumbnail is generated from the video. Information about the video, such as duration, date and time of the recording and the quality used, is also saved. Additionally, time stamps for each recorded frame are stored in a text file.

4.3 Filtering

When the user enters the stabilization screen (figure 4.4), the sensor data is read back into the program and estimates of the orientation of the device are calculated according to algorithm 1 in section 2.2.5. The reference row quaternions are then low-pass filtered before the rectification and stabilization starts, according to the amount of stabilization that the user has chosen.

Since gyroscope and accelerometer measurements are not received at the same time instances, the accelerometer values are interpolated to the gyroscope time.
4.4 Rectification and Stabilization

The methods for image rectification and video stabilization from sections 2.3.2 and 2.3.3 have been implemented in the application. The camera calibration matrix $K$ (section 2.1.3) was estimated for different resolutions from videos of a calibration pattern by using OpenCV [12].

The application relies on OpenGL ES to do the actual rectification and stabilization. This means that the graphics hardware can be utilized for calculations and for drawing the frames of the new video.

4.4.1 OpenGL ES

The rectification and stabilization process for each frame follows five steps:

1. Read the frame from the recorded video.
2. Send the image data to OpenGL ES together with orientation estimates.
3. Draw the new, rectified and stabilized, frame.
4. Get the resulting image data from OpenGL ES.
5. Append the new frame to the rectified and stabilized video.

Reconnecting to section 3.3.1, one could note that the orientation estimates are treated as uniform variables in OpenGL ES, as they stay the same when rendering a single frame. The same holds for the camera calibration matrix and its inverse, which are also sent to the graphics card.

Step 3 involves creating a mesh of vertices, which represents the points for which we will calculate new, rectified and stabilized positions. Figure 4.6 shows a mesh overlaid on the frame of a video. The figure just serves as an illustration; a much denser grid is used in the application. Post-rectification, the mesh has been transformed as in figure 4.7. The pixel values of the output image are interpolated from the surrounding vertices, much like the example in section 3.3.2. Vertex positions are used as attribute variables in OpenGL ES. The texture position associated with a vertex is a varying type variable.

The rectification and stabilization can, as seen in figure 4.7, lead to black borders in the resulting image. This issue can be addressed using two different methods, or a combination of the two. The first method is using the graphics hardware’s capabilities to extrapolate data. Just by increasing the size of the mesh, OpenGL ES will do its best to fill the previously black areas. The other method is cropping or zooming the frame to remove the black borders. Figures 4.8 and 4.9 show the
Figure 4.6. A vertex mesh overlaid on a frame.

Figure 4.7. The vertex mesh after rectification.
Figure 4.8. Rectified and stabilized frame without extrapolation.

same video frame, with and without extrapolation. No zoom or cropping were used.

Figure 4.9. Rectified and stabilized frame with extrapolation.
Chapter 5

Evaluation

This chapter starts with the presentation of a user study, where the methods of rectification and stabilization from chapter 2 were compared to those of competing applications in a blind experiment. Conclusions are drawn from the results, with charts to help visualize the collected data.

In the second section, the results of the thesis are discussed. The strengths and weaknesses of using inertial measurement sensors for video stabilization and rolling shutter correction are pointed out. The chapter ends with some ideas of future work in the area.

5.1 User Study

A user study was conducted to evaluate the video stabilization and rectification algorithms that were implemented on the iPod Touch 4. Two video sequences, approximately ten seconds long, were recorded with the iPod. The sequences were then processed by four different applications: Deshaker [3], iMovie ’11 [2], Movie Stiller [17] and the iPhone/iPod application from chapter 4.

The first sequence was captured while walking along the campus street of Linköping University and was intended mainly to test the stabilization capabilities of the applications. This clip has lots of shaking and a number of moving objects (people). Figure 5.1 shows three frames from this video sequence.

The second sequence is similar to the first but also contains fast panning to get some noticeable rolling shutter distortion. It features fewer moving objects. This sequence will test performance of both stabilization and rectification algorithms. Three frames from the sequence are shown in figure 5.2. In the figure’s middle frame one can see rolling shutter effects.

The settings for each application were chosen in a way that let them use their
Figure 5.1. Frames 1 (top), 137 (middle) and 274 (bottom) of the first evaluation video.
Figure 5.2. Frames 1 (top), 185 (middle) and 280 (bottom) of the second evaluation video.
particular strengths. For example, if an application supports some kind of edge-filling, one can usually allow a little less zoom and still get good results. Similarly, the adaptive filtering technique used in the iPhone/iPod application reduces the size of the black borders during fast motion and so it can use a smaller zoom value than other applications while maintaining a greater amount of stabilization where there are small changes in camera orientation.

In general, the zoom values were chosen high enough to remove most of the black edges, but not excessively high since this causes the videos to lose sharpness.

5.1.1 Deshaker

Deshaker is a video stabilizer plugin for the open source video processing utility VirtualDub [18]. In addition to stabilization, Deshaker can also do rolling shutter correction and supports edge compensation where black borders in a frame are filled with information from previous and future frames.

Deshaker version 2.7 was used to process the videos and most of the settings were left at their default values. The most important parameter changes was setting the rolling shutter “amount” to 72% and enabling Deshaker to use previous and future frames to fill borders. A fixed zoom of 10% was used for the first video sequence and 20% fixed zoom was used for the second. This was to avoid too large borders. Even though Deshaker’s edge-filling methods work well in general, moving objects can cause some strange effects when interpolating between frames.

5.1.2 iMovie ’11

iMovie is a video editing software created by Apple. iMovie has been able to stabilize videos since the ’09 version and new to the version of 2011 is a rolling shutter fix. iMovie ’11 does not offer many settings and is not able to perform any edge filling. Instead, the amount of stabilization is chosen by setting a maximum zoom value. The amount of rolling shutter correction can be chosen as “None”, “Low”, “Medium”, “High” or “Extra High”. A zoom of 27% and Extra High rolling shutter correction were used for both video clips that were processed by iMovie ’11.

5.1.3 Movie Stiller

Movie Stiller is an application for the iPhone, iPod and iPad that can stabilize movies recorded on the device. In contrast to the other application used in the user study, it does not correct for rolling shutter distortions. There are two parameters that a user of Movie Stiller can choose: Stabilization Strength and Default Scale. Setting a large stabilization strength without increasing the scale would then lead
5.1 User Study

to black borders in the resulting video, because Movie Stiller does not support any edge-filling.

To get a fair comparison against iMovie '11, a zoom value of 27% was used in both videos and the stabilization strength was tuned so that little or no borders were visible in the output videos. The stabilization strength was set to 3.5 and 0.3, respectively. Version 1.1 of Movie Stiller was used to do the stabilization.

5.1.4 The iPhone/iPod Application

For stabilization, the iPhone/iPod application used an adaptive low-pass filter with minimum filter size $N = 29$ and maximum filter size $N = 99$ (as described in section 2.3.3). The zoom was set to 15% and 25%, respectively. The mesh size was increased so that any black borders remaining were filled with extrapolated image data (see section 4.4.1).

5.1.5 Method

The user study was conducted as a blind experiment, where users were shown pairs of videos and asked to choose the one they thought looked the best. They had no prior knowledge about what software was used or even what the applications tried to achieve (i.e. a stable video without rolling shutter distortion).

There were five versions of each of the two recorded video sequences: the original recording and the output of the four applications described above. All versions were compared to each other, meaning that a user would make a total of 20 decisions for the two movies. The order in which the versions appeared was randomized, but was the same for all users.

The form used in the study had a concluding question asking if the user would consider using a video stabilization application on their mobile phone. The form (in Swedish) is attached in appendix A.

5.1.6 Ranking

From the ten comparisons for a video sequence, we want to rank the five versions of the clip. The versions are awarded one point for each “win” against another version. The ranking is then based on the number of points that the versions have collected. Versions with the same number of points are considered equal. For example, if versions A and B got three points each, C got two points and D and E got one point each, then A and B would be ranked first, C would be placed third and D and E would share the fourth position.
30 people, aged 18 to 36, participated in the user study. Figure 5.3 shows the total number of points for each version of the first video sequence. As seen, the iPhone/iPod application got the highest number of points, followed by Deshaker. iMovie ‘11 and Movie Stiller performed similarly, and the original video had the least number of wins against other versions.

From this one can draw the conclusion, that it is certainly possible to improve upon the original video sequence and that in general, people preferred the stabilized versions to the original one. Not all users agreed though, as can be seen in figure 5.4(a). A few of the participants ranked the original first, possibly because it was the only one not suffering from the bluriness caused by zooming. In figure 5.4 we also see that the iPhone/iPod application got by far the most first placements.

The total number of points for each version of the second video sequence are displayed in figure 5.5. The iPhone/iPod application scored the highest number of points also for this sequence, with Deshaker not far behind. Despite the lack of rolling shutter correction, Movie Stiller got the third most points. The original video sequence outperformed iMovie ‘11, which got the least amount of points.

The increased number of points for the original video (in comparison to the first video sequence) indicates that this second video sequence was more difficult to stabilize and rectify. Still, the iPhone/iPod application and Deshaker performed well, whereas iMovie ‘11 failed to improve upon the original video.
Figure 5.4. Histograms of the rank of each version of the first evaluation video.
Figure 5.5. The total number of points for each version of the second evaluation video.

Figure 5.6 shows that the iPhone/iPod application was ranked first the most number of times, just as with the first video sequence. 5 people thought that none of the applications gave a better result than the original video, as seen in 5.6(a). The loss in image quality due to the large zoom levels can be one reason for this. Also, the quick panning seems to have made things more difficult, at least for iMovie '11.

Whether the rolling shutter distortions affected the results is hard to tell, since there was no version of the video sequence that was corrected only for RS distortion, but not stabilized. Movie Stiller did get more points than iMovie ’11 however, so a successful stabilization is more important than correcting the RS distortion (assuming, of course, that iMovie ’11 managed to corrected this distortion).

The question of whether the user would consider using video stabilization software on their mobile phone was answered affirmative by 28 out of the 30 participants. One person did not answer the question. This shows that there is an interest among users to be able to improve the quality of their recorded videos. How much they would be willing to pay for such a video stabilization application was not examined. Time and battery consumption could also be important factors to the users.

To sum up, the methods of video stabilization and rolling shutter correction presented in the thesis performs equal to or better than the algorithms implemented in Deshaker, iMovie ’11 and Movie Stiller, at least when applied to the two video sequences used in the study. We can also see that the output of the iPhone/iPod application clearly improves over the original video sequences. One should remem-
Figure 5.6. Histograms of the rank of each version of the second evaluation video.
ber though, that the iPhone/iPod application has access to data that the other applications do not, namely the accelerometer and gyroscope measurements.

5.2 Discussion

The problem studied in the thesis has been that of correcting distortions caused by motion of a handheld video-recording device with a rolling shutter camera, utilizing sensor data from accelerometer and gyroscope. Methods to solve this problem have been presented, and implemented as an iPhone/iPod application. The video stabilization and image rectification capabilities of the application have then been compared to those of competing software.

The purpose of the thesis, which was to “implement and evaluate a video stabilization algorithm and a rolling shutter correction algorithm on a mobile video-recording device using gyroscope and accelerometer data to track the orientation of the device”, has thus been fulfilled. The rectification method presented in [4] has successfully been modified to work with orientation estimates calculated from sensor data, making it possible to avoid solving computationally demanding optimization problems.

Another advantage of relying on inertial measurement sensors instead of image processing techniques is that it does not matter what is recorded, so scenes with, for example, many moving objects or poor light conditions is not a problem. Note though, that image processing may still be necessary to synchronize the sensor data to the recorded frames.

The major weakness of this sensor-based approach to stabilization and rectification is that it requires a gyroscope and an accelerometer to be attached to the camera. The sensors need to have high enough update rates and accuracy, so that orientation estimates of reasonable quality can be extracted. Also, using sensors introduces the problem of synchronizing the recorded video sequences to the sensor data. Processing a video that is not synchronized to the gyroscope and accelerometer measurements gives a very poor result, so this issue must be treated with care.

The conducted user study shows that the rectification and stabilization algorithms can be implemented on current mobile, video-recording devices with good results. The iPhone/iPod application performs equal to or better than Deshaker, iMovie ’11 and Movie Stiller when applied to the two videos used in the study.

The goal of the thesis, to create an iPhone/iPod application able to record videos and then rectify and stabilize them has, in part, been reached. While the most important functionality is there, the application is not yet ready to be released for common use.
5.3 Future Work

There are a number of areas in which one could continue the work of the thesis. Using gyroscopes and accelerometers with higher update rates than those of the iPod Touch 4, it should be possible to improve the quality of rectification by utilizing more than one orientation estimate for each frame, using piece-wise interpolation to obtain the orientation of a specific row from these estimates.

Future work on the topic of the thesis could also include expanding the motion model so that also translations are taken into account. To find the position of the device the accelerometer data would have to be integrated two times, making the impact of measurement noise large. Still, one could possibly obtain estimates that are reasonably accurate during short time intervals, like the readout time of a camera. One reason not to include translations in the model is that a pure rotational model is presented as the best option in [4].

Using sensor fusion to combine the measurements from gyroscope and accelerometer with the information from a KLT tracker would be another way to improve the quality of rectification and stabilization, at the cost of increased time consumption. One could also use the orientation estimates for other purposes, such as de-blurring the frames of a video, similar to what is described in [5].
Bibliography


Appendix A

User Study Form

Inledning


**Film 1**

<table>
<thead>
<tr>
<th>Filmpar</th>
<th>Vänster</th>
<th>Höger</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Film 2

<table>
<thead>
<tr>
<th>Filmar</th>
<th>Vänster</th>
<th>Höger</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Avslutning

Filmera du såg var två originalinspelningar samt fyra videostabiliserade versioner av vardera film. Skulle du kunna tänka dig att använda en applikation som utför stabilisering på din mobiltelefon?

☐ Ja         ☐ Nej

Du är _____ år gammal.
Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämd som upphovman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/

© Gustav Hanning