Improving Image Stabilization in Modern Surveillance Cameras

Emil Manelius

September 2023

Abstract

This thesis investigates potential ways to improve the image stabilization in Axis Communications surveillance cameras. The current in-house developed stabilization algorithm, Axis EIS, stabilizes based on gyroscope measurements. We compare a purely gyroscope based stabilization algorithm with a gyroscope-accelerometer sensor fusion stabilization algorithm, image-based stabilization algorithms and stabilization algorithms combining image analysis and gyroscope measurements. Footage from surveillance cameras are stabilized using these different approaches and the algorithms are evaluated based on image quality as well as their implementation viability in a real-time stabilization pipeline running on the camera. The results indicate that a purely gyroscope-based stabilization algorithm is difficult to surpass in a real-time environment due to its good qualitative performance relative to its low computational complexity. Simple image-based techniques could complement a gyro-based stabilizer if sufficient feature point tracking speed is attained.

Acknowledgements

First of all, I would like to thank my suprevisors Tor Nilsson, Johan Förberg, Ivar Persson and Magnus Oskarsson for their invaluable advice and feedback throughout this project. I would also like to thank everyone at Core Tech Graphics at Axis for always offering their help whenever needed, in ways both big and small. Finally, I would like to thank my friends and family for their unrelenting support and encouragement during this time.

Contents

Chapter 1

Introduction

Image stabilization refers to the process of removing the effects of unwanted camera motion from a recorded video. For example, a hand-held mobile phone recording a video will look shaky since the photographers hands are not 100% still. The result is a video that is jittery and hard for a viewer to follow. To rectify this, image stabilization is applied to remove the effects of the shaking, producing a smooth video. Image stabilization is not only useful for hand-held devices, but an important feature in modern surveillance cameras as well. Cameras may for example be mounted in tall masts to get good overview of critical areas. These masts, like most tall objects, will shake when exposed to heavy wind with a shaky surveillance feed as a consequence. Axis Communications has its own proprietary algorithm for image stabilization, referred to as Axis EIS. This algorithm performs well in most cases, but struggles in certain difficult conditions, such as very high zoom settings which is a common case for mast-mounted cameras.

1.1 Purpose and research question

The goal of this thesis project is to investigate potential alternatives to the current image stabilization algorithm, Axis EIS. Ultimately, this thesis hopes to present an algorithm that provides an improvement over Axis EIS whilst also being viable for implementation in a modern surveillance camera. Particularly, the goal is to cover up some of the common issues stabilization algorithms similar to Axis EIS struggle with, such as stabilization under high zoom, without sacrificing the quality in the situations where the current stabilizer performs well.

Is it possible to develop a real-time image stabilization algorithm that outperforms the current Axis EIS?

In order to answer this question, many different approaches to image stabilization have been investigated. The approaches that have seemed promising have been implemented and analyzed with our goals in mind. Since Axis EIS is proprietary software, it cannot be described here in full detail. Because of this, we will instead make the comparison with another algorithm that for the purposes of this thesis can be considered equivalent to Axis EIS (see Section 2.2 for details).

1.2 Requirements for a successful solution

If one wishes to improve current Axis EIS, criteria for improvement needs to be established. Most importantly, a candidate algorithm must provide qualitatively better stabilization than Axis EIS. The algorithm performs very well in many cases while failing in certain reproducible scenarios, particularly when the camera is subjected to heavy zoom and very slight vibrations. The pragmatic goal here is to produce an algorithm that offers similar stabilization levels to Axis EIS when it works as intended, and better in scenarios where it fails. Another important requirement is real-time performance. Axis EIS stabilizes frames as they come in without extra delay in the image pipeline. A camera typically operates around 30 FPS, so a stabilization algorithm should ideally stabilize incoming frames in an online manner at this rate. An algorithm can be computationally more complex than the current Axis EIS, as long as it can run at full frame rate on a camera. Finally, robustness is required. For the surveillance use case, it is important that the image is never significantly distorted by the stabilization algorithm in degenerate cases.

1.3 Problem model and limitations

The goal of image stabilization is to remove video artifacts stemming from unwanted camera motion. A shaking camera will mainly produce two kinds of artifacts: frame-to-frame jitter and rolling shutter wobble.

If a shaking camera captures a static scene, the image coordinates of the same static scene point will be displaced between consecutive frames, making the video harder to follow. Most modern image sensors, including the ones used in Axis cameras, capture the image line by line. As a result, a shaking camera will produce frames that look wobbly, since the camera has moved not only between consecutive frames, but also between individual scan lines within the same frame. A good stabilization algorithm needs to remove both of these artifacts from the video feed.

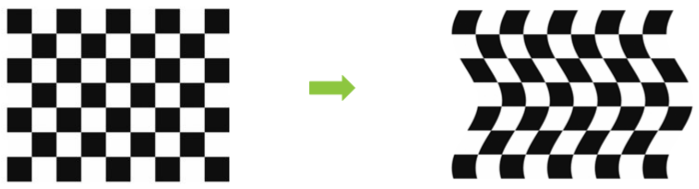

Figure 1.1: Illustration of rolling shutter artifacts. If a camera captures the checkerboard in the left image whilst moving periodically back and forth horizontally, capturing one line at a time from the top down, the resulting image will look similar to the right image

For a surveillance camera, most motion is unwanted since the camera is normally mounted in a fixed location to monitor a fixed scene. We thus want to rectify the video to give the video the appearance of being captured with a motionless camera, in contrast to for example the hand-held camera stabilization case where one seeks to smooth out a wanted motion trajectory. Pan-Tilt-Zoom (PTZ) cameras have motors that can adjust their viewing direction and zoom levels. We will not be considering image stabilization during PTZ-movement.

1.4 Types of motion

The typical case for a shaking camera is a camera mounted in an exposed location, for example a tall camera mast, which subjects it to external forces such as wind that cause shaking. The resulting motion is thus typically periodic about some fixed center. The motion can be divided into translational motion and rotational motion. Since cameras typically observe far away scenes, the effects of the rotational motion is significantly more prevalent in the video feed, and are what is most important to correct for to achieve good stabilization. The types of stabilization algorithms that will be considered either attempts to stabilize in the image-space directly, in which case the camera motion model is less important, or algorithms that attempt to model camera rotations.

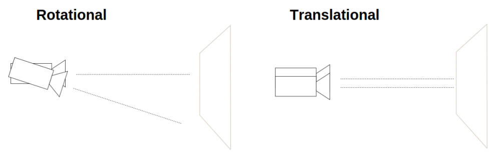

Figure 1.2: Illustration of the difference between translational and rotational movement. A small angular displacements leads to large pixel displacements in the final image for far-away scenes, whereas translational motion does not scale in the same way

Chapter 2

Background & Related work

2.1 A few concepts from computer vision

To lay the foundation necessary to discuss methods for image stabilization, we need to introduce some concepts from computer vision. This section is based on the lecture notes by Carl Olsson [1] and if the reader feels familiar with this topic this section can be disregarded. Computer vision deals with the problem of reconstructing a 3D-scene based on images of said scene. In the context of image stabilization, we consider a camera capturing two consecutive frames I and J of the same real world scene, having moved slightly between the two frames. The task is to use the information in the images I and J to estimate the movement of the camera between them and applying a transform to J that undoes the effect of this movement.

2.1.1 The pinhole camera model

To describe how a camera projects a 3D-scene into a 2D image, we introduce the pinhole camera model, which describes this projection for an ideal pinhole camera. The pinhole camera is a light proof box with a tiny apperture (the pinhole) that light can enter through. This will produce a projected image of the scene on the back of the camera wall (see Figure 2.1).

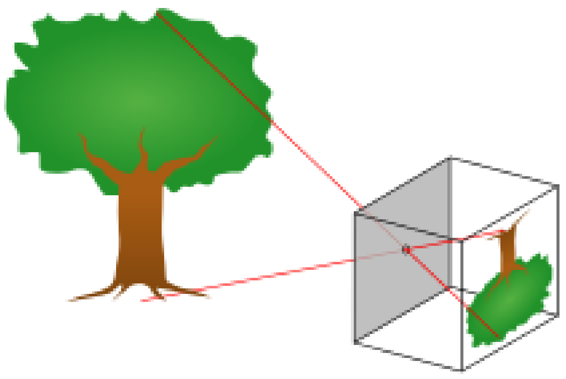

Figure 2.1: The pinhole camera (image taken from Wikimedia Commons[2])

To set up the mathematical model for this projection, we begin by introducing a 3D coordinate system such that the pinhole of our camera, C, is located at the origin. To get the projection x of a point in the world $X = (X_1', X_2', X_3')$, we form the line between the pinhole and the real world point, called the viewing ray. The directional vector of the line is $X - C$ and the line intersects C, so we get the parametrisation

$$l : C + s(X - C) = sX, s \in \mathbb{R} \tag{2.1}$$

since C is located at the origin. To get the projection x, we find the intersection point between the viewing ray and the image plane, which we can define as the plane $z = 1$ in our coordinate system. Thus, the projection will satisfy $sX_3' = 1 \implies s = \frac{1}{X_3'}$, meaning we get

$$x = \begin{bmatrix} \frac{X_1'}{X_3'} \\ \frac{X_1'}{X_3'} \\ 1 \end{bmatrix}. \tag{2.2}$$

So, to get the projection of a real world point X we divide by the third coordinate in the coordinate system that places the pinhole at the origin. The image coordinates will then be given by the first and second coordinates. To do image stabilization, we need to consider cameras that have moved relative to each other. Assume the point X has coordinates that are known in some global reference system, (X_1, X_2, X_3). To get the projection of the point in our pinhole camera, we apply a transform that moves the camera to the origin and rotates it to look down the z axis, i.e. for some rotation matrix R and some translation vector t, we have

$$\begin{bmatrix} X_1' \\ X_2' \\ X_3' \end{bmatrix} = R \begin{bmatrix} X_1' \\ X_2' \\ X_3' \end{bmatrix} + t. \tag{2.3}$$

With our imposed limitations, we generally assume that cameras are only rotated relative to each other, so the vector t can, for our purposes, be set to 0. With this model, the image we get is centered at the origin and measured in some 3D length unit. To finalize the model, we need to define a mapping from this ideal image plane to pixel space, where the origin is typically located in the top left corner and the pixel resolution can vary for the same scene depending on camera settings. This mapping is defined by a matrix containing the so called inner parameters and has the form

$$K = \begin{bmatrix} \gamma f & sf & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.4}$$

Here, f is the focal length, γ the aspect ratio, s the skew and (x_0, y_0) the image center in pixel space. So, to summarize, the camera model, the model that maps a real world point X to an image point x is given by

$$\lambda \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = KR \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \tag{2.5}$$

where K represents internal camera parameters and R the orientation of the camera. To get the projection, one multiplies KR with X and normalizes the result on the third coordinate. We will hereafter use $x \sim KRX$ to denote the operation of carrying out the matrix multiplication KRX and dividing by the third coordinate to get the projection.

2.1.2 Corresponding points

With our camera model in place, we can discuss corresponding points. Consider two images I and J captured by the cameras $P_1 = K_1R_1$ and $P_2 = K_2R_2$ respectively (we consider the same real world camera with different rotations as two different cameras) and some 3D point X. This $3D$ point will have be projected onto the point $x \sim P_1X$ in image I and the point $y \sim P_2X$ in image J. The pair (x, y) are called corresponding points: points in two different image corresponding to the same real world point.

2.1.3 Homographies

If x and y are two different image points

$$x = \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}, y = \begin{pmatrix} y_1 \\ y_2 \\ 1 \end{pmatrix} \tag{2.6}$$

one can be interested in finding invertible transforms that map point y to point x. In our computer vision framework, these transforms can be represented as 3-by-3 matrices, such that

$$x \sim Hy. \tag{2.7}$$

Such transforms are typically called homographies.

2.1.4 Feature points

In order to undo the effects of camera motion between two frames I and J, one typically attempts to find a homography that maps points in the image J to points in the image I. This often requires a large number of known pairs of corresponding points. Finding such pairs presents a challenge. The way this is done is often by finding the location of key feature points in the image. Feature points are points of distinct features in an image, such as the corners of a building, that are easily identifiable in different images of the same object. To get correspondences, feature points are either estimated in two different images and then matched according to some criterion, or tracked from one image to the next based on their locations in the previous image. Algorithms for feature point detection and tracking are discussed in more detail in Section 2.4.8.

2.1.5 RANSAC

In the coming sections, we will deal with problems with this general outline: Assume that K corresponding points x_k, y_k have been found and we want to fit a model f based on these pairs with the goal of minimizing some cost function C:

$$\hat{f} = \min_f \sum_{k=1}^{K} C(f(x_k, y_k)). \tag{2.8}$$

In general, not all pairs of corresponding points we find will be true corresponding points, since the algorithms for producing these estimates are not exact. This means that we will have a situation where a few estimated point correspondences produce extreme outliers, making least squares estimation of the model an unsuitable choice. Instead, random sample consensus, RANSAC, can be used. The RANSAC procedure is as follows:

1. Draw a minimal number of point correspondences required to fit f randomly and fit f using only these

2. Evaluate $C(f(x_k, y_k))$ for all pairs (x_k, y_k). If the cost is below some predefined threshold, a pair is considered an inlier, else an outlier. Let the inliers form the consensus set.

3. If the consensus set is bigger than the previously largest consensus set, save f and the consensus set

4. Repeat 1 - 3 a predefined number of times. The estimated f with the largest consensus set is taken as the solution.

2.2 Gyro-based stabilization

Axis EIS uses gyroscope-based image stabilization, similar to what is used by Karpenko et al.[3], which we will use as a placeholder for the Axis EIS algorithm. For the purposes of this thesis, these two algorithms can be considered equivalent. Karpenko et al. [3] assumes that the camera is only displaced rotationally, for similar reasons as discussed earlier. They use the following model for the projection x of a world point $X = (X_1, X_2, X_3)$ at time t:

$$x = KR(t)X. \tag{2.9}$$

This means that the projection of X at times t_n and t_{n+1} are related by

$$x_{t_n} = KR(t_n)R^T(t_{n+1}))K^{-1}x_{t_{n+1}}. \tag{2.10}$$

The gyroscope measures angular velocity $\omega = (\omega_x, \omega_y, \omega_z)$ in the local coordinate system of the gyroscope, which we assume is aligned with the coordinate system of the camera. If the rotation is known at time point t_n, the rotation at time point t_{n+1} can be obtained by compounding changes in rotation between consecutive gyro samples. We assume that the angular velocity $\omega = (\omega_x, \omega_y \omega_z)$ of the camera between two consecutive samples is constant. This means that the angular displacement about each axis between timepoints t_n and t_{n+1}, called $\Delta\phi_{t_n}$, can be approximated with

$$\Delta\phi_{t_n} = \omega_{t_n}\Delta t, \tag{2.11}$$

where $\Delta t = t_{n+1} - t_n$ [3]. The camera has three axes about which it can be rotated: roll, pitch and yaw (see figure 2.2). Rotation about the viewing axis, roll, will result in a pixel displacement in the image that does not scale with the viewing distance.

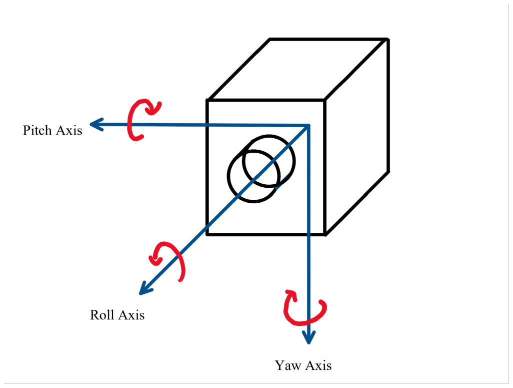

Figure 2.2: Roll, pitch and yaw rotations

For similar reasons as the earlier discussed translational displacements, these rotations do not produce significant artifact and are therefore ignored. If the change in angle between times t_n and t_{n+1} is small and roll angle is ignored, the relation in Equation (2.10) can be approximated as a translation in pixel space, proportional to the angular displacement in yaw and pitch direction (Equation (11) in [3], see figure 2.3 for an illustration).

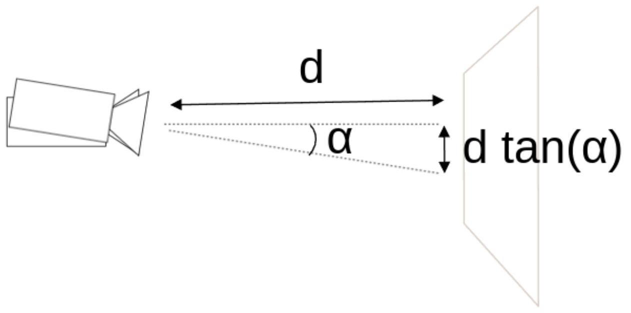

Figure 2.3: If the angle α is small enough, we get the approximation $d\tan(\alpha) \approx d\alpha$. Here, d is the distance to the image plane, given by the focal length. See section 2.1 for details.

Thus, the per-pixel displacement of the image since some reference time t_0 with known rotation can be approximated as proportional to the sum of the angular velocities since that reference time:

$$\Delta x_n \propto \sum_{k=0}^{n} \omega_{x_{t_n}}, \Delta y_n \propto \sum_{k=0}^{n} \omega_{y_{t_n}}. \tag{2.12}$$

To account for accumulating error, we add a decay factor θ, s.t. $0 < \theta < 1$ to the discrete integrator, yielding

$$\Delta x_n \propto \sum_{k=0}^{n} \theta^{n-k} \omega_{x_{t_n}}, \Delta y_n \propto \sum_{k=0}^{n} \theta^{n-k} \omega_{y_{t_n}}. \tag{2.13}$$

Karpenko et al. estimate the rotations with the gyroscope in the manner outlined, then smoothen the estimated rotations with a low-pass filter. The image is then stabilized at time t with the relative displacement between the smoothed and raw rotation at that timepoint. Since we assume that the camera should not move from its initial orientation, we calculate the per-pixel displacement since the start time with Equation (2.13) and then stabilize the image by moving the frame at timestep t_n a pixel distance $-x_n, -y_n$. This model is easy to adapt to account for rolling shutter. Under rolling shutter, the capture time of a point depends on its y-coordinate in pixel space. This means that the pixel space displacement of the image will be different for each line. Similar to Karpenko et. al, we divide the image horizontally into a number of segments and displace each segment according to the capture time of the current line. Gyroscopes typically have some small constant bias, ω_b, affecting the measurements. To deal with this, the raw gyro samples are fed through a high pass filter before integration. We use a discrete time high pass filter of the form

$$u_n = \beta u_{n-1} + \frac{1+\beta}{2}(\omega_n - \omega_{n-1}) \tag{2.14}$$

where ω is the input gyro signal $\omega = (\omega_x, \omega_y, \omega_z)$ and u the filtered samples.

2.3 Gyroscope-Accelerometer sensor fusion

The previous approach relies purely on the gyroscope to measure the camera displacements. The gyroscope is part of an intertial measurement unit (IMU) that also contains an accelerometer, which could potentially be incorporated into a gyro-based stabilization algorithm. The accelerometer measures translational acceleration. In theory, one could use the accelerometer to estimate the translational movements of the camera and incorporate that into the model for the camera motion by integrating the measurement values twice (given appropriate initial conditions), as discussed in for example Karpenko et al.[3]. They draw the conclusion that this is not feasible in practice since the sensor is not accurate enough to produce reliable results after double integration. Even if this was not the case, adding translations to the model has limited value to begin with as discused earlier. It is however possible to use the accelerometer to obtain measurements of camera orientation. An accelerometer at rest will measure the force of gravity in the local coordinate system of the IMU (which again is assumed to be the same as the coordinate system of the camera, since the IMU is mounted close to the image sensor). One can then calculate the rotation that would rotate the estimate of the gravity vector in a global coordinate frame to the estimated gravity vector in the local coordinate frame. The advantage of this is that it provides an instantaneous measurement of camera orientation. The gyroscope measures orientation relative to the previous orientation, whereas the accelerometer measures orientation relative to a fixed global coordinate system, making it less susceptible to accumulation of errors, and providing an additional source of information. Combining gyroscope and accelerometer measurements to estimate camera orientation can be accomplished with a Kalman filter, as done by for example Trawny et al. [4], Solá [5] or Mirzaei et al. [6]. We will use a similar Kalman filter set-up as the one derived in Trawny et al.[4].

2.3.1 The error state Kalman filter

The Kalman filter produces the optimal linear reconstruction, interpolation and prediction of the state vector for a dynamic system, given observations of inputs and outputs to this system [7]. The standard Kalman filter is the optimal linear estimator for a linear system. The state model for the Kalman filter looks as follows: given a state space vector at time t, x_t, the system is described as evolving in time according to

$$x_{t+1} = Fx_t + e_t \tag{2.15}$$

$$z_t = Hx_t + w_t \tag{2.16}$$

where x_t represents the process' internal state and the vector z_t an observation of the true state x according to the measurement model described by the matrix H, e_t represents the process noise and w_t the measurement noise, here assumed to be zero-mean Gaussian noise with covariance matrices Q_t and R_t respectively.

Assuming that we have predictions and observations of the state up to time point t, the Kalman filter predicts the state for the time point $t+1$ and the covariance matrix $P_{t+1|t}$ in the following manner.

$$\hat{x}_{t+1|t} = F_t \hat{x}_{t|t}. \tag{2.17}$$

$$P_{t+1|t} = F_t P_{t+1|t} F_t^T + Q_t \tag{2.18}$$

and then refines that prediction using measurements taken at time step $t+1$ to get the posterior estimates. First, calculate the residual between the measurement and the predicted measurement using the estimate of the state so far and the measurement model

$$\tilde{y}_{t+1} = z_{t+1} - H_{t+1}\hat{x}_{t+1|t}. \tag{2.19}$$

The Kalman gain K_{t+1} is calculated according to

$$S_{t+1} = H_{t+1} P_{t+1|t} H_{t+1}^T + R_t \tag{2.20}$$

$$K_{t+1} = P_{t+1|t} H_{t+1}^T S_{t+1}^{-1} \tag{2.21}$$

and then used together with the residual to update our estimate of the state vector and the covariance matrix

$$\hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + K_{t+1}\tilde{y}_{t+1} \tag{2.22}$$

$$P_{t+1|t+1} = (I - K_{t+1}H_{t+1})P_{t+1|t} \tag{2.23}$$

In general, the state transition and the observation model does not need to be linear functions of the state. If they instead are differentiable functions of the state, one can choose to linearize around a current estimate. This is usually referred to as the extended Kalman filter. If the state transition is defined as $x_{t+1} = f(x_t) + e_t$ for some differentiable function f and the measurement model as $z_t = h(x_t) + w_t$, one gets the extended Kalman Filter estimates by replacing H and F with the Jacobians of h and f respectively in the update step, and otherwise proceeding as usual. The extended Kalman filter will in general not be optimal, since it approximates the true system dynamics with a first order Taylor expansion. To perform well, this requires the state to stay close to the point around which the expansion is done. To increase the likelihood that this approximation stays valid, one can instead track the error state with the Kalman filter, the difference between the true state and some nominal state. The idea is that the nominal state is a large signal and the error state small, making the linear approximations more accurate for the error state. In our case, the nominal state would be the integrated gyro samples and our current approximation of the bias in the gyroscope. The error state consists of the error in orientation from our current nominal orientation estimate, and the current gyro bias error. The challenge is now to describe and linearize the system dynamics model for the error-state.

2.3.2 Quaternions to represent rotations

To lay the necessary foundation for the state-space model derivation, we must first discuss some basic quaternion algebra and how to represent rotations in three dimensions with quaternions. Quaternions are an extension of the complex numbers and are defined as

$$q = q_4 + q_1 i + q_2 j + q_3 k \tag{2.24}$$

with the hyperimaginary units i, j, k satisfying the following identities

$$i^2 = j^2 = k^2 = -1, -ij = ji = k, -jk = kj = i, -ki = ik = j. \tag{2.25}$$

The quaternion can also be written on matrix form:

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}. \tag{2.26}$$

Worth pointing out here is that there are many different quaternion conventions, differing in for instance whether $ji = k$ or $ji = -k$ [5]. The derivations in [4], that we are basing our filter model on, use JPL convention which is what we use here as well. A quaternion is often referred to as having a scalar part, q_4 and a vector part, $\mathbf{q} = (q_1, q_2, q_3)$ in our definition. If these satisfy

$$\mathbf{q} = \begin{bmatrix} k_x \sin(\theta/2) \\ k_y \sin(\theta/2) \\ k_z \sin(\theta/2) \end{bmatrix} = \hat{\mathbf{k}} \sin(\theta/2), q_4 = \cos(\theta/2) \tag{2.27}$$

with $\hat{\mathbf{k}}$ having unit length, q is called a rotation quaternion and encodes a rotation in 3 dimensions along the axis $\hat{k}$ by the angle θ. Quaternion multiplication between the quaternions q and p, hereafter denoted $q * p$ can be defined by the matrix multiplication

$$q * p = \begin{bmatrix} p_4 & -p_3 & p_2 & p_1 \\ p_3 & p_4 & -p_1 & p_2 \\ -p_2 & p_1 & p_4 & p_3 \\ -p_1 & -p_2 & -p_3 & p_4 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}. \tag{2.28}$$

Note that this does not in general commute. If q and p represents two rotation quaternions, their product represents a composition of the rotations. With this multiplication definition, we get the neutral element $q_0 = 1 + 0i + 0j + 0k$ satisfying

$$q * q_0 = q_0 * q = q. \tag{2.29}$$

For rotation quaternions, we get the multiplicative inverse

$$q^{-1} = \begin{bmatrix} -\mathbf{q} \\ q_4 \end{bmatrix} \tag{2.30}$$

satisfying $q * q^{-1} = q^{-1} * q = q_0$. Another useful way to write the quaternion multiplication on matrix form is

$$q * p = \begin{bmatrix} p_4 I_{3\times3} + \lfloor \mathbf{p} \rfloor_\times & \mathbf{p} \\ -\mathbf{p}^T & p_4 \end{bmatrix} \begin{bmatrix} \mathbf{q} \\ q_4 \end{bmatrix} \tag{2.31}$$

where

$$\lfloor \mathbf{p} \rfloor_\times = \begin{bmatrix} 0 & -p_3 & p_2 \\ p_3 & 0 & -p_1 \\ -p_2 & p_1 & 0 \end{bmatrix} \tag{2.32}$$

which is the matrix for the operator of taking the cross product with p: $p \times v = \lfloor \mathbf{p} \rfloor_\times v$. A useful result as a consequence of this is that, since $u \times v = -v \times u$

$$\lfloor u \rfloor_\times v = -\lfloor v \rfloor_\times u \tag{2.33}$$

Another matrix we will use is

$$\Omega(\omega) = \begin{bmatrix} -\lfloor \omega \rfloor_\times & \omega \\ \omega^T & 0 \end{bmatrix}. \tag{2.34}$$

Trawny et al. [4] gives the following formula to relate the rotation quaternion to the corresponding rotation matrix (see [4] for full derivation)

$$C(q) = (2q_4^2 - 1)I_{3\times3} - 2q_4 \lfloor \mathbf{q} \rfloor_\times + 2\mathbf{q}\mathbf{q}^{\mathbf{T}}. \tag{2.35}$$

If the rotation angle $\delta\theta$ is small, we can make the approximations $\sin(\delta\theta/2)) = \delta\theta/2, \cos(\delta\theta/2) = 1$, yielding the approximation

$$q \approx \begin{bmatrix} \frac{1}{2}\delta\theta \\ 1 \end{bmatrix} \tag{2.36}$$

and

$$C(q) \approx I_{3\times3} - \lfloor \delta\theta \rfloor_\times. \tag{2.37}$$

We are now ready to derive the time derivative for the quaternion:

$$\dot{q}(t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t}(q(t+\delta) - q(t)). \tag{2.38}$$

We can define δq as

$$q(t+\delta t) = \delta q * q(t) \tag{2.39}$$

where δq represents the rotation aligning $q(t)$ with $q(t+\Delta t)$. When Δt approaches 0, this rotation becomes very small, and we can apply the approximation from (2.36) to δq, yielding

$$\dot{q}(t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t}(q(t+\delta) - q(t)) = \lim_{\Delta t \to 0} \frac{1}{\Delta t}(\delta q * q - q), \tag{2.40}$$

which we can approximate by

$$\dot{q}(t) \approx \lim_{\Delta t \to 0} \frac{1}{\Delta t}(\begin{bmatrix} \frac{1}{2}\delta\theta \\ 1 \end{bmatrix} - \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix})q = \lim_{\Delta t \to 0} \frac{1}{\Delta t}(\begin{bmatrix} \frac{1}{2\Delta t}\delta\theta \\ 0 \end{bmatrix}., \tag{2.41}$$

We notice that

$$\omega = \lim_{\Delta t \to 0} \frac{\delta\theta}{\Delta t} \tag{2.42}$$

is the angular velocity. Applying this, and the definition of the quaternion product, we get

$$\dot{q} = \frac{1}{2}\begin{bmatrix} \omega \\ 0 \end{bmatrix} * q = \frac{1}{2}\begin{bmatrix} -\lfloor \omega \rfloor_\times & \omega \\ \omega^T & 0 \end{bmatrix} q = \frac{1}{2}\Omega(\omega)q. \tag{2.43}$$

If we assume that the rotational velocity remains constant over a time step Δt, we can integrate the quaternion over the time step Δt by solving the first order differential equation in 2.43. This is a known problem with the matrix exponential as solution, so we get

$$q(t+\Delta t) = \exp(\frac{1}{2}\Omega(\omega)\Delta t)q(t). \tag{2.44}$$

2.3.3 Updating the nominal state

We now have the necessary background to continue discussing the Kalman filter setup. The first step is to describe how to update our nominal state. The system state is determined by the estimated orientation and the estimated gyroscope bias. Since the bias is assumed to change slowly, this is simply updated as $\hat{b}_{t+1} = \hat{b}_t$. The estimated orientation is updated from time step t to $t+1$ by integrating the bias-corrected gyroscope measurements over the time step Δt, according to Equation (2.44)[4]. We thus get the nominal state at time point $t+1$, u_{t+1} as

$$u_{t+1} = \begin{bmatrix} \exp(\frac{1}{2}\Omega(\hat{\omega})\Delta t)\hat{q}_t \\ \hat{b}_t \end{bmatrix} \tag{2.45}$$

where

$$\hat{\omega} = \omega - \hat{b}_t \tag{2.46}$$

is the bias-corrected measurement.

2.3.4 Deriving the state-space representation for the error state

Trawny et al. [4] derive the state-space representation for the error state for an IMU-based kalman filter (a similar derivation can be found in for instance [5] as well). The necessary parts of the derivation are outlined in the following section. We model the gyro as measuring the true angular velocity of the IMU, along with some bias and some noise, here assumed to be Gaussian white noise with zero mean and constant variance:

$$\omega_m = \omega + b_g + n_g, n_g \in \mathcal{N}(0, \sigma_g). \tag{2.47}$$

Similar assumptions are made for the accelerometer measurements:

$$a_m = a + b_a + n_a, n_a \in \mathcal{N}(0, \sigma_a). \tag{2.48}$$

The gyro bias is assumed to drift slowly over time. For our state vector, we use

$$\mathbf{x} = [\delta q^T, \quad \Delta b_g^T]. \tag{2.49}$$

Here, δq is the error-quaternion (as a column vector) and Δb_g is the gyro bias error. The error quaternion represents the small rotation required to rotate our initial estimate of the IMU orientation with the true orientation of the IMU:

$$\delta q \approx \begin{bmatrix} \frac{\delta\theta}{2} \\ 1 \end{bmatrix} \tag{2.50}$$

such that

$$q = \delta q * \hat{q}. \tag{2.51}$$

The gyro bias error represents the difference between the true gyro bias and our initial estimate of the gyro bias, With the state vector in place, we are ready to find our state transition model. We begin by linearzing the continuous time state equations for the error quaternion. We take the time derivative of equation 2.51:

$$\dot{q} = \dot{\delta q} * \hat{q} + \delta q * \dot{\hat{q}}. \tag{2.52}$$

Using the result on the quaternion derivate (Equation (2.43), we get

$$\frac{1}{2}\begin{bmatrix} \omega \\ 0 \end{bmatrix} * q = \dot{\delta q} * \hat{q} + \delta q * (\frac{1}{2}\begin{bmatrix} \hat{\omega} \\ 0 \end{bmatrix} * \hat{q}) \tag{2.53}$$

Rewriting yields

$$\dot{\delta q} * \hat{q} = \frac{1}{2}(\begin{bmatrix} \omega \\ 0 \end{bmatrix} * q - \delta q * \begin{bmatrix} \hat{\omega} \\ 0 \end{bmatrix} * \hat{q}). \tag{2.54}$$

Multiplying both sides by $\hat{q}^{-1}$ from the right yields

$$\dot{\delta q} = \frac{1}{2}(\begin{bmatrix} \omega \\ 0 \end{bmatrix} * \delta q - \delta q * \begin{bmatrix} \hat{\omega} \\ 0 \end{bmatrix}). \tag{2.55}$$

Now, by our model for the gyro measurements, we have

$$\omega = \hat{\omega} - \Delta b - n_r. \tag{2.56}$$

Substituting this into 2.55 we get

$$\dot{\delta q} = \frac{1}{2}(\begin{bmatrix}\hat{\omega}\\0\end{bmatrix} * \delta q - \delta q * \begin{bmatrix}\hat{\omega}\\0\end{bmatrix}) - \frac{1}{2}\begin{bmatrix}\Delta b + n_r\\0\end{bmatrix} * \delta q. \tag{2.57}$$

Using the multiplication form from (2.31), we get

$$\dot{\delta q} = \frac{1}{2}(\begin{bmatrix}-\lfloor\hat{\omega}\rfloor_\times & \hat{\omega}\\ -\hat{\omega}^T & 0\end{bmatrix}\delta q - \begin{bmatrix}\lfloor\hat{\omega}\rfloor_\times & \hat{\omega}\\ -\hat{\omega}^T & 0\end{bmatrix}\delta q) - \frac{1}{2}\begin{bmatrix}\Delta b + n_r\\0\end{bmatrix} * \delta q \tag{2.58}$$

simplifying to

$$\dot{\delta q} = \frac{1}{2}(\begin{bmatrix}-2\lfloor\hat{\omega}\rfloor_\times & 0_{3\times1}\\ -0_{1\times3} & 0\end{bmatrix}\delta q) - \frac{1}{2}\begin{bmatrix}-\lfloor(\Delta b + n_r)\rfloor_\times & (\Delta b + n_r)\\ (\Delta b + n_r)^T & 0\end{bmatrix}\begin{bmatrix}\frac{\delta\theta}{2}\\1\end{bmatrix}, \tag{2.59}$$

which means

$$\dot{\delta q} = \frac{1}{2}(\begin{bmatrix}-2\lfloor\hat{\omega}\rfloor_\times & 0_{3\times1}\\ -0_{1\times3} & 0\end{bmatrix}\delta q) - \frac{1}{2}\begin{bmatrix}\Delta b + n_r\\0\end{bmatrix} - O(|\Delta b||\delta\theta|, |n_r||\delta\theta|). \tag{2.60}$$

Since the errors and bias variance are small, we can make the approximation

$$\dot{\delta q} = \begin{bmatrix}-\hat{\omega}\times\delta\theta - \frac{1}{2}(\Delta b + n_r)\\0\end{bmatrix} \tag{2.61}$$

meaning we get

$$\dot{\delta\theta} = -\hat{\omega}\times\delta\theta - \frac{1}{2}(\Delta b + n_r). \tag{2.62}$$

For the bias, we have

$$\dot{\Delta b} = n_w \tag{2.63}$$

so we finally arrive at the linearized continuous time state equations for the error vector:

$$\begin{bmatrix}\dot{\delta\theta}\\ \dot{\Delta b}\end{bmatrix} = \begin{bmatrix}-\lfloor\hat{\omega}\rfloor_\times & -I_{3\times3}\\ 0_{3\times3} & 0_{3\times3}\end{bmatrix}\begin{bmatrix}\delta\theta\\ \Delta b\end{bmatrix} + \begin{bmatrix}-I_{3\times3} & 0_{3\times3}\\ 0_{3\times3} & I_{3\times3}\end{bmatrix}\begin{bmatrix}n_r\\ n_w\end{bmatrix} = F\begin{bmatrix}\delta\theta\\ \Delta b\end{bmatrix} + G\begin{bmatrix}n_r\\ n_w\end{bmatrix}. \tag{2.64}$$

To be able to use this with our discretely sampled IMU measurements, the continuous time state model must be discretized. If we assume that the system dynamics matrix F remains constant over a small time step Δt (corresponding to constant angular velocity rotation during this time interval), we can get the state transition matrix between the discreet time steps t and $t+\Delta t$, $\Phi(t+\Delta t, t)$, is given by

$$\Phi(t+\Delta t, t) = \exp(F\Delta t). \tag{2.65}$$

This is in turn approximated by the first order Taylor expansion of, yielding

$$\hat{\Phi}(t+\Delta t, t) = I_{6\times6} + F\Delta t. \tag{2.66}$$

This is the state transition matrix that we use for the state transition and update step in the extended Kalman filter.

The observation model

What remains is to derive the observation model matrix, H. Trawny et al. [4] use a sun-sensor instead of the accelerometer measurements to get an estimate of the current orientation, but the derivation will be similar for the accelerometer. Since the camera is assumed to only be rotating, this will measure the force of gravity in the local coordinate system of the IMU. If the matrix C represents the rotation of the global frame to the local IMU frame, the accelerometer will measure

$$a = Cg + n_a \tag{2.67}$$

where g is the force of gravity in the global coordinate system. We now need to relate the state vector to the measurement error. The measurement error, $\tilde{z}$ fulfills

$$\tilde{z} = z - \hat{z} = (C - \hat{C})g + n_a. \tag{2.68}$$

By definition of our error quaternion, we have

$$C(q) = C(\delta q)C(\hat{q}). \tag{2.69}$$

Additionally, since the rotation error is small, we have

$$C(\delta q) \approx I - \lfloor \delta\theta \rfloor_\times \tag{2.70}$$

Putting this into equation 2.68 we get

$$\tilde{z} = (I - \lfloor \delta\theta \rfloor_\times - I)C(\hat{q})g + n_a = -\lfloor \delta\theta \rfloor_\times C(\hat{q})g + n_a. \tag{2.71}$$

By Equation (2.33) we then get

$$\tilde{z} = \lfloor C(\hat{q})g \rfloor_\times \delta\theta + n_a \tag{2.72}$$

meaning

$$\tilde{z} = \begin{bmatrix} \lfloor C(\hat{q})g \rfloor_\times & 0_{3\times3} \end{bmatrix} \begin{bmatrix} \delta\theta \\ \delta b \end{bmatrix} + n_a \tag{2.73}$$

meaning we have the observation matrix

$$H = \begin{bmatrix} \lfloor C(\hat{q})g \rfloor_\times & 0_{3\times3} \end{bmatrix}. \tag{2.74}$$

Applying the error state estimate to the nominal state

With the error state Kalman filter formulation we have chosen, the error vector will contain an error quaternion relative to the current nominal state. After performing the filter update step, the nominal state needs to be updated, and the error state reset. If the nominal state is given by

$$u = \begin{bmatrix} \hat{q} \\ \hat{b} \end{bmatrix} \tag{2.75}$$

and the error state

$$x = \begin{bmatrix} \delta\hat{q} \\ \Delta\hat{b} \end{bmatrix} \tag{2.76}$$

we get the updated nominal step through

$$u_{new} = \begin{bmatrix} \delta\hat{q} * \hat{q} \\ \hat{b} - \Delta\hat{b} \end{bmatrix}. \tag{2.77}$$

After applying this update, the error-state is set to zero since we have applied all our knowledge of the current error to the nominal state. We also need to update our covariance matrix P accordingly to reflect this change in uncertainty. If the error state update function is g, such that

$$x_{reset} = g(x), \tag{2.78}$$

we need to update P as

$$P_{new} = GPG^T \tag{2.79}$$

where G is the Jacobian of g. This will be a 6×6 matrix where the top $3x3$ part corresponds to applying the inverse rotation of $\delta\hat{q}$ and the bottom part to removing the bias $\Delta\hat{b}$. The bottom part is simply the identity matrix. The top 3×3 part will be $I - \frac{1}{2}\delta\hat{\theta}$, which for small errors can be approximated as just identity. Thus, we simplify to

$$P_{new} = P \tag{2.80}$$

2.3.5 Summary of the error state Kalman filter

To summarize, we get the following procedure for updating the camera orientation estimate from time step t to $t+1$ with a Kalman filter combining gyroscope and accelerometer data:

1. Update the nominal state as described in Section 2.3.3.
2. Propagate the error state and covariance according to equations (2.17) and (2.18).
3. Update the error state according to equations (2.19) through (2.23).
4. Update the nominal state with the posterior error state.
5. Apply the error state reset.

2.4 Image analysis-based image stabilization

Gyro-based stabilization algorithms struggles in certain cases to distinguish between sensor noise and real camera movement. If the camera movement impacts the video feed, the information about that movement is by definition available in the images. This suggests that stabilization methods based on image analysis might be a good alternative in cases where gyro-based methods perform poorly, such as under high zoom. Much prior work has been done on image analysis-based image stabilization. Typically, the focus is on smoothing out a jittery camera path captured by for example a hand-held mobile phone, such as in Grundmann et al. [8] or Greicher & Liu [9]. These methods are offline and smooth the entire video as a post-processing step, where we instead need real time stabilization.

2.4.1 Fitting a homography between frames

If we ignore rolling shutter and consider consecutive images I and J, where the goal is to undo the effects of camera motion between them, we can use standard computer vision methods (see for example Chapter 5 in [1] for more details) to fit a homography between the images I and J if we have an estimated set of corresponding points between the images. The image can then be stabilized by applying the inverse homography to J, mapping points in J to the corresponding points in I and undoing the motion. Finding these homographies involve solving a linear system of equations (typically systems as found in Equations (5.27) and Equation (5.28) in [1]). If we allow ourselves to use the same assumptions as for the gyro-based stabilizer, with no roll and small angular displacements, we can get a computationally cheaper model.

2.4.2 Global motion estimation

The simplest model one could use to achieve image stabilization is to fit a global motion vector between consecutive frames. Under the assumptions made by [3], the small camera rotation in pitch and yaw direction will result in corresponding points begin related by $x_k+(dx,dy) = y_k$ for some constant displacement dx, dy. If x_k, y_k is a pair of corresponding points in two consecutive frames, dx, dy are found such that

$$\sum_{k=0}^{n} d(x_k + (dx, dy), y_k)^2 \tag{2.81}$$

is minimized. The second frame is then translated by $(-dx, -dy)$ to get the stabilized frame. Since the feature estimations are going to contain outliers, RANSAC can be used instead of least-squares minimization to get more robust results. This approach does a good job of removing frame-to-frame jitter, but we do not currently handle rolling shutter artifacts at all.

2.4.3 Rolling shutter correction

Rolling shutter correction is a challenge for image-based stabilization algorithms. The main issue is that the artifact affects the image in such a way that the transformation can no longer be described by a homography. The simplest approach one could use for image based rolling shutter correction is to segment the image along the y-axis and apply the same approach as described in the global motion estimation section to each segment independently instead of to the entire image at once. This comes with a heavy requirement on having many feature points divided evenly across the entire image. Another problem with this approach is that it can fail quite spectacularly. If one segment happens to contain many outliers it can become severely distorted compared to the rest of the image. Much work has been done on image-based rolling shutter correction. Forssén and Ringaby [10] use an approach where they fit a rotation spline model for the camera motion between frames from a set of point correspondences, which we will discuss in more detail.

2.4.4 Rotation Spline

The approach used by Forssén and Ringaby to do rolling shutter correction is to fit a rotation spline between consecutive frames. First, the intrinsic camera matrix K is assumed to be known. They also assume that the camera is only rotating about the camera center, so the model for the projection x of a world point X at time t is given by

$$x \sim KR(t)X. \tag{2.82}$$

We denote the time it takes for the image sensor to capture a single image row t_r and the frame rate f. The time between the capture of the last row in frame n and the first row of frame $n+1$ is denoted t_d and the number of rows in an image N_r. The current time can, for convenience, be expressed in terms of number of rows, where one ”row” time unit is equal to t_r. The delay between two frames can be expressed in a number of blank rows N_b, such that

$$N_b = N_r t_d f = Nr(1 - t_r f). \tag{2.83}$$

Consider a pair of corresponding points

$$x = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ 1 \end{bmatrix} \tag{2.84}$$

in two consecutive frames with real world coordinates X. They will satisfy

$$x \sim KR(N_x)X, y \sim KR(N_y)X \tag{2.85}$$

where N_x, N_y represents the capture time for point x and y respectively, where $N_x = x_2$ and $N_y = N_r + N_b + y_2$ (with time zero at the start of the capture of the first of the two frames). We then get the relation

$$x \sim KR(N_x)R^T(N_y)K^{-1}y = Hy. \tag{2.86}$$

If no constraint is made on the rotations $R(N_x)$, $R(N_y)$, each correspondence that is not on the same row as previous correspondences introduces 6 new unknowns (since rotations can be parameterized with a minimum of 3 unknowns).

This is solved by modeling the rotation between the two frames as a spline with M knots, each knot being a rotation at a fixed time point that needs to be estimated. To get the intermediate rotations between the fixed notes, spherical linear interpolation (SLERP) is used. SLERP corresponds to assuming that the rotation between R_m and R_{m+1} has constant angular velocity and fixed axis of rotation.

2.4.5 Parametrization and interpolation of rotations

The rotations are parameterized as vectors $n \in \mathbb{R}^3$, where $n = \phi\hat{n}$ such that ϕ is the rotation angle and $\hat{n}$ is the axis of rotation ($||\hat{n}|| = 1$). Rodrigues formula can be used to convert this representation to a rotation matrix:

$$R = \text{expm}(n) = I + \lfloor \hat{n} \rfloor_\times + \lfloor \hat{n} \rfloor_\times^2 (1 - \cos\phi) \tag{2.87}$$

where the cross product matrix is the same as in the quaternion section. To convert back, the formula

$$n = \text{logm}(R) = \tan^{-1}(||\tilde{n}||, trR - 1)\frac{\tilde{n}}{||\tilde{n}||} \tag{2.88}$$

with

$$\tilde{n} = \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}. \tag{2.89}$$

For details, see the original paper by Forssén and Ringaby [10]. They use this to define the spherical linear interpolation between two rotations n_1 and n_2 using the interpolation parameter $\tau \in [0, 1]$ as

$$n_{diff} = \text{logm}(\text{expm}(-n_1)\text{expm}(n_2)) \tag{2.90}$$

$$R_\tau = \text{expm}(n_1)\text{expm}(\tau n_{diff}) \tag{2.91}$$

2.4.6 Finding the homography between the frames

With the camera motion model in place, we need to fit an appropriate rotation spline based on a set of observed corresponding points. If we have K matching points between frames, the goal is to estimate the M knot rotations $n_1, ..., n_M$ for the rotation spline that minimizes the loss function

$$J = \sum_{k=1}^{K} d(x_k, Hy_k)^2, \tag{2.92}$$

with H defined as in Equation (2.86). The distance function d is given by

$$d(x, y) = (x_1/x_3 - y_1/y_3)^2 + (x_2/x_3 - y_2/y_3)^2. \tag{2.93}$$

Let N_m be the row time for spline knot number m. To find $R(x_k) = R(N_x)$ needed to compute H, one calculates

$$\tau = \frac{N_x - N_m}{N_m + 1 - N_m}, N_, \leq N_x \leq N_{m+1} \tag{2.94}$$

and then gets

$$R(N_x) = SLERP(n_m, n_{m+1}, \tau) \tag{2.95}$$

where SLERP is defined by equations (2.90) and (2.91). Using rotations that are relative to the first frame of the current frame pair, the first key rotation can be initialized to $(1, 0, 0)^T$, yielding $3(M - 1)$ degrees of freedom for the estimation of H.

2.4.7 General homography mixture

Grundmann et al. expanded on the approach used by Forsen and Ringaby in [11] and introduced a calibration-free version of their rolling shutter correction algorithm. The rotation spline essentially represents a combination of M different homographies that are fit to different parts of the image. Grundmann et. al skipped the intermediary step of estimating the camera rotation and fit the homographies directly. Instead of segmenting the image into separate blocks (done before, get source), they use gaussian weights centered on various image segments to get smooth interpolation between the homographies in different parts of the image. They also introduce a regularization term to handle cases where a large segment of the image has very few feature points. Since we can estimate the camera matrix K in our use case (since we would know which camera an algorithm will run on), we will stick to the approach used by Forssén and Ringaby, but it is worth keeping in mind that there are alternatives if K is hard to obtain.

2.4.8 Estimating and tracking feature points

All of the above mentioned image stabilization methods rely heavily on feature point estimation and matching. Given the performance constraints for the intended application, it is important that feature points can be extracted and matched efficiently. Scale-Invariant Feature Transform, SIFT [12], is a popular method for feature point estimation. SIFT is designed to match features between images through heavy distortions, begin invariant to changes in scale and image rotation. An example use case would be finding a book on a table given a close-up image of the cover. While SIFT is very robust and generally produces matches with high accuracy, it is computationally costly. Rublee et. al [13] suggest the ORB (Oriented FAST and Rotated BRIEF) feature point estimation and matching algorithm as a more efficient alternative to SIFT. ORB operates in a similar way to SIFT, but uses a less expensive descriptor set that preserves the same invariants as the descriptor set originally used in SIFT. According to the authors, ORB offers approximately two orders of magnitude better performance than the standard SIFT. For our use-case however, simpler methods may suffice. SIFT (and ORB) pay substantial computational costs to ensure that features can be tracked through heavy image distortions. Under our assumptions, the distortions between consecutive frames comes from a slight camera rotation in a camera monitoring a far-away scene, meaning that the frames have similar scales and are not significantly rotated relative to each other. Under these conditions, given a set of feature points in an image I, they can be tracked into the next image J instead of having to estimate new features in J and then matching the features between I and J. We will use the problem formulation and algorithm described by Bouguet [14], which is an implementation of the algorithm originally developed by Lucas and Kanade [15]. In Bouguet[14] the tracking problem is formulated as follows: given a feature u in the image I and a subsequent image J, we want to find the displacement vector d such that $I(u_x, u_y)$ and $J(u_x + d_x, u_y + d_y)$ are, in some sense, similar. More precisely, we want to find d such that

$$\epsilon(d) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y+w_y}^{u_y+w_y} (I(u_x, u_y) - J(u_x + d_x, u_y + d_y))^2 \tag{2.96}$$

is minimized for some integration window size w_x, w_y. The displacement vector d is sometimes referred to as the optical flow vector, it describes how a certain pixel moves into the next image.

2.4.9 Pyramidal Feature Tracking

We now describe the algorithm from [14] that is used to solve the tracking problem. At the minima, the derivative of ϵ in equation 2.96 will be the zero vector. We have

$$\frac{\partial \epsilon(d)}{\partial d} = -2 \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y+w_y}^{u_y+w_y} (I(u_x, u_y) - J(u_x + d_x, u_y + d_y)) \cdot \begin{bmatrix} \frac{\partial J}{\partial x} & \frac{\partial J}{\partial y} \end{bmatrix} \tag{2.97}$$

If the displacement vector $d = (d_x, d_y)$ is small, we can approximate $J(u_x + d_x, u_y + d_y)$ by the first-order Taylor expansion around d, $J(u_x + d_x, u_y + d_y) \approx J(u_x, u_y) + [\frac{\partial J}{\partial x} \quad \frac{\partial J}{\partial y}][d_x \quad d_y]^T$. Here, the algorithm uses the assumption that the image J is similar to the image I in a neighborhood around u. If these images only differ by a small pixel-wise translation (which generally will hold under our assumptions of small rotational camera displacements and a far-away scene), we get the approximation

$$[\frac{\partial J}{\partial x} \quad \frac{\partial J}{\partial y}]^T \approx [\frac{\partial I}{\partial x} \quad \frac{\partial I}{\partial y}]^T = [I_x \quad I_y]^T = \nabla I. \tag{2.98}$$

Inserting all this, we get

$$\frac{\partial \epsilon(d)}{\partial d} = -2 \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y+w_y}^{u_y+w_y} (I(x, y) - J(x, y) - \nabla I^T d) \nabla I^T. \tag{2.99}$$

Notice that $I(x,y) - J(x,y)$ represents a kind of time-derivative for the image I. Introducing $\delta I = I(x,y) - J(x,y)$, we get

$$\frac{\partial \epsilon(d)}{\partial d} \approx 2 \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y+w_y}^{u_y+w_y} (\nabla I^T d - \delta I) \nabla I^T \tag{2.100}$$

Writing out the matrix multiplication yields

$$\frac{1}{2}\frac{\partial \epsilon(d)}{\partial d} \approx \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y+w_y}^{u_y+w_y} (\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} d - \begin{bmatrix} \delta I I_x \\ \delta I I_y \end{bmatrix}). \tag{2.101}$$

Now introduce

$$G = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y+w_y}^{u_y+w_y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{2.102}$$

and

$$b = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y+w_y}^{u_y+w_y} \begin{bmatrix} \delta I I_x \\ \delta I I_y \end{bmatrix}). \tag{2.103}$$

We get

$$\frac{1}{2}\frac{\partial \epsilon(d)}{\partial d} \approx Gd - b \tag{2.104}$$

so the d minimizing ϵ will be $d = G^{-1}b$. An important assumption here was that the displacement vector is small, so the first order Taylor approximation is valid. The iterative Lucas-Kanade optical flow algorithm therefore does many iterations of this computation to get better accuracy. Introduce the initial guess $d^0 = [d_x^0, d_y^0 \quad]^t$. Then, if the displacement d^{k-1} at iteration level $k-1$ is known (and J^k denotes J shifted by d^{k-1}), we get $d_k = G^{-1}b^k$, where

$$b = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y+w_y}^{u_y+w_y} \begin{bmatrix} \delta I_k I_x \\ \delta I_k I_y \end{bmatrix}). \tag{2.105}$$

and

$$\delta I_k(x,y) = I(x,y) - J_k(x,y). \tag{2.106}$$

This can be repeated for a fixed number of iterations, or until convergence. The pyramidal version operates by creating a pyramid of downscaled images, using the iterative Lucas-Kanade scheme to calculate the optical flow at each pyramidal level, feeding the solution from the previous pyramidal layer as a starting guess for the next layer. When this scheme is used, the approximation of small displacements has a much higher chance of being valid. A large pixel displacement in the initial image will be small further down in the pyramid.

2.4.10 Good features to track

The question of how to get the initial feature point estimate still remains. Of course, one could use SIFT or ORB to get an initial estimate of the feature points, but this might not be optimal given that those algorithms are not necessarily optimized for tracking. Instead, we use the approach described by Shi and Tomasi in [16]. In brief, features are found by finding the points in the image where the matrix G in Equation (2.102) satisfies

$$\min(\lambda_1, \lambda_2) \geq \epsilon \tag{2.107}$$

where ϵ is some predefined threshold and λ_1, λ_2 are the eigenvalues of G. Their reasoning is that these kinds of features are easily tracked by the previously described tracking algorithm.

2.5 Background-Foreground segmentation

A common issue for stabilization algorithms that run on feature point tracking is the difficulty in separating image movement due to camera motion and image movement from moving objects in the scene. This is partly addressed by using outlier rejection when fitting the motion model to the tracked features. However, if one has an accurate model for detecting moving objects in a scene, this could be improved further since these points could be masked out before fitting the model in the first place. This problem is called background-foreground segmentation and there are many different models available. For our use-case it is important to keep the real-time constraint in mind. Maintaining a background model adds significant computational overhead making a cheaper model preferable to a more expensive one. Fortunately, since it is only used to improve the robustness of the feature point tracking over time and not in the actual stabilization calculation, it does not have to be updated on every frame, as long as the updates are frequent enough to capture movement of typical large real life objects, which gives some leeway. Another important constraint to keep in mind is that the background model should behave reasonably in scenarios where the stabilization algorithm fails.

2.5.1 Pixel difference

The simplest background-foreground segmentation algorithm one could use is to take the pixel difference between consecutive stabilized frames and apply some threshold. This has the advantage of being very fast, but does not perform well when the stabilization is not perfect. If there is shakiness remaining in the video, edges of stable objects will be considered moving and masked out, which is highly undesirable since those are the kinds of regions most suitable for feature point tracking in the first place. This risks a scenario where a temporary failure of the stabilization cascades.

2.5.2 Dense optical flow

Another way to obtain a background-foreground mask is to use dense optical flow. Dense optical flow is the estimation of the displacement vector in Equation (2.96) for every pixel in the image. The first step is to take some estimate of the dense optical flow between two consecutive frames. Then, one can calculate the mean optical flow in the image. A camera moving between frames will result in a global motion vector for each pixel. For each pixel, the distance between the dense optical flow vector and the median optical flow vector is calculated and thresholded. This is in many ways similar to the pixel difference approach, but more robust to stabilization failures, at the cost of an expensive dense optical flow estimation. It is worth pointing out here that one could do image stabilization with dense optical flow in this way, by estimating the inter-frame motion as the median dense optical flow vector. This would be too expensive to do on every frame however, but can more realistically be done in the background model which can be updated more sparsely. Since the foreground mask does not have to be pixel perfect, the dense optical flow estimation can also be done on a downscaled version of the image, since we only need to capture relatively large movements without the higher accuracy requirements one would have if this was the basis for the stabilization algorithms itself.

2.6 Hybrid methods

Of course, the choice between using IMU data and image analysis for stabilization is not mutually exclusive. Ideally, these two approaches can be combined into a hybrid stabilization algorithm if both data sources are available to use. For instance, [6] and [17] use a Kalman filter to combine camera motion estimates from a set of corresponding points and the IMU. Shi et al. used deep learning to combine dense optical flow estimates with gyroscope data to do image stabilization for a mobile phone [18]. They use a neural network to produce dense optical flow estimates on downsampled video frames, which are fed to a CNN downsampling it to 4 output channels. The output from this network is concatenated with gyroscope data represented as rotation quaternions. The resulting time series is fed through an LTSM, which produces an orientation estimate for the camera. We will however primarily focus on a more straightforward approach to combining the different stabilization approaches.

2.6.1 Multi-pass stabilization

One way to combine IMU-based stabilization with image based stabilization is to do multiple stabilization passes. When a frame is read, an IMU-based stabilization pass is performed on the image, which is then fed to an image-based stabilization algorithm that adds an extra correction layer to the frame. This has the very attractive property of addressing the problem at hand directly. If the IMU-based stabilizer fails to stabilize the image for some reason, the image based stabilization steps in to correct. The obvious downside to this is that such an approach is very expensive. The full image needs to be corrected and transformed twice, instead of only applying a single transformation.

2.6.2 Efficient multi-pass stabilization

Fortunately, a two-pass stabilization approach can be sped up significantly by merging the two passes into a single one, if the second pass consists of a stabilization step that fits a model to a sparse feature set. Such algorithms do not need to estimate the features on the stabilized frames. Instead, one can estimate and track the features on the raw input frames and then apply the sensor based image transform to the feature set before fitting the model. Additionally, one can use the information from the IMU-based stabilization step to improve the feature point tracking in the unstabilized frame. By applying the inverse stabilization transform from the IMU-based stabilizer one can produce a guess as to where the feature points are supposed to have moved in the new frames. These two ideas combined leads to feature point tracking that is almost equivalent to tracking on IMU-stabilized frames, without actually having to apply the transform to the full image in between. We will now discuss how this can be done using the purely gyroscope-based stabilizer described in Section 2.2.

2.6.3 Stabilizing the feature points

Assume that we have some feature set x_k from the frame I that has been tracked into the next frame J yielding the corresponding points y_k and some estimate of the location of the corresponding points to $\hat{x_k}$ in $\hat{I}$, where $\hat{I}$ is I stabilized with the gyro stabilizer. To get the estimation of the corresponding points $\hat{y}_k$ in the gyro-stabilized J, called $\hat{J}$, we just subtract the offsets calculated with Equation (2.13) from y_k, yielding

$$\hat{y}_k = y_k - (\Delta x, \Delta y). \tag{2.108}$$

2.6.4 Producing the IMU-based initial guess for the new feature point locations

We would like to use the inverse of the gyro-based stabilization transform to produce a better initial guess for the feature point locations, to get better feature point tracking accuracy and performance. If the camera had a global shutter, this would be trivial. In that case, one could simply calculate an initial guess for $\hat{y}_k$, $\hat{\hat{y}}_k$ with

$$\hat{\hat{y}}_k = x_k + (\Delta x, \Delta y) \tag{2.109}$$

(with $(\Delta x, \Delta y)$ calculated with Equation (2.13)). Under rolling shutter however, things are slightly more complicated. The offset for the image points is a function of time, $(dx, dy) = \Delta(t)$, since the camera moves continuously between the frames. The estimated new location for the point x_k is

$$\hat{y_k} = x_k + \Delta(t_{c_k}) \tag{2.110}$$

where t_{c_k} is the capture time of the point x_k. This capture time is in turn dependent on the offset itself, since each row in the image is captured at a different time point. The function $\Delta(t)$ is sampled discreetly at each gyro measurement. To get $\hat{y_k}$ one can iterate through the samples of $\Delta(t)$ and for each sample time t_s check if $t_c(x_k + \Delta(t_s)) \geq t_s$. The first time this happens, the point will have been captured, and $t_s \approx t_{c_k}$. Iterating through every sample of $\Delta(t)$ for every tracked feature point is inefficient, but since $\Delta(t)$ is the same for every point and image rows are captured from the bottom and up, we know that the point with the smallest y coordinate will be captured first. One can therefore maintain a queue of feature points sorted by y coordinates, and check the capture-time condition for the lowermost point each time a new gyro sample comes in, popping the points from the queue and appending them to a vector containing $\hat{\hat{y}}_k$ as the condition is met. With this setup, producing the gyro-based guess for the feature point locations in the next frame adds very little overhead to the stabilization process.

2.7 Quantifying image stability

To properly compare different stabilization algorithms, it is important to have some metrics to evaluate against. For this purpose, three different metrics are used: Peak-Signal-to-Noise ratio, structural similarity between consecutive frames and information loss. The first three estimate how similar consecutive frames look, whereas information loss is important to measure to ensure that an algorithm actually preserves a large portion of the image.

2.7.1 PSNR

Given an $m \times n$ image I and a noisy approximation of that image K, one can define PSNR as

$$PSNR = 10 \log_{10}(\frac{MAX_I^2}{MSE}) \tag{2.111}$$

where MAX_I is the maximum pixel value in the image I and MSE is the mean squared error between the image I and the approximation K:

$$MSE = \sum_{i=1}^{m} \sum_{j=1}^{n} (I(i,j) - K(i,j))^2. \tag{2.112}$$

To measure image stabilization, one can consider the frame f_{n+1} an approximation of the previous frame f_n. A video feed with higher average PSNR across all consecutive frame pairs can then be considered more stable then one with lower average PSNR.

2.7.2 Structural similarity

Structural similarity is a perception-based model of similarity between two images [19]. The structural similarity is defined as a multiplicative combination of a luminance term, a contrast term and a structural term. Let x and y denote two equally sized subwindows in the images I and K. The structural similarity between these windows is then

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{2.113}$$

(Equation (13) in [19]) where μ_x, μ_y denotes the mean pixel value in the window x and y respectively, σ_x^2, σ_y^2 the variance in each window and σ_{xy} the cross-covariance between the windows. C_1 and C_2 are regularization parameters defined as $C_1 = (K_1L)^2$ and $C_2 = (K_2L)^2$. K_1 and K_2 are design parameters typically set to some value smaller than one and L is the dynamic range of the images. The rationale behind the measurement is that noise in the image is less noticeable in regions that are either very bright or very textured. To get the score between two full frames, the measure is averaged across each window pair.

2.7.3 Information loss

Most digital image stabilization algorithms require some cropping of the video. In order to penalize excessive cropping, we also measure information loss for each stabilized video. To define this, the minimal crop required to get a frame without black regions is calculated. The measure is then the ratio between the cropped video size and the full frame size for each frame in the feed. A similar metric was used in for instance [18].

Chapter 3

Method

3.1 Data

The following section contains information about the data that is used to evaluate the stabilizers.

3.1.1 Video sets

Several videos, along with appropriate sensor data, have been recorded with two different Axis provided PTZ cameras and are used to test the image stabilization algorithms. The first camera is a model P5655 PTZ camera which may record videos at 25 FPS and 1920x1080 resolution. The second camera is a Q6318 model PTZ camera. This camera records at 30 FPS and has a sensor resolution of 3820x2160, but the video material is recorded at 1920x1080 resolution for easier comparison. Gyroscope and accelerometer data is recorded with a sample rate of 500 hz. The cameras both have different IMU sensors, with the IMU in the Q6318 being slightly more accurate. Five video sets are used for evaluation. The cameras are set up in a camera mount loosely fitted to a wooden base, that can be subjected to manual shaking. Five (include real world set later) different data sets are recorded and used to test the various image-based stabilization algorithms. Each data set contains a video with varying degrees of shakiness and an accompanying gyroscope and accelerometer series. The first set, P5655_road_zoom, is a recording of a road crossing with light traffic and some flag poles waving in the wind in the foreground. This is recorded at full zoom for the P5655, 31x magnification. The second set is Q6318_road_zoom_31. This set is recorded with the Q6318 camera at 31x magnification. The scene recorded is a road with light traffic and surrounding buildings. The third set is Q6318_building_zoom_31. This is recorded with the Q6318 camera at 31x magnification. The scene is a building wall and contains very little foreground movement. The fourth set, Q6318_door_zoom_15 is a recording of a door by the Q6318 camera at 15x magnification. The final set, Q6318_table_zoom is a recording of a table and accompanying chairs at 15x zoom with the Q6318 camera. Each set gives the algorithms 300 consecutive frames worth of data to update any internal state with before stabilization begins to ensure appropriate settling time. After this, the video is stabilized for 600 consecutive frames. The sensor-fusion Kalman filter does not have any scene dependence. This can therefore be evaluated on a simpler scene than the image based stabilizers. Two additional sets are recorded with the Q6318 camera mounted in the same position. The camera records a checkerboard pattern located on the wall, at no zoom. In the first of these two sets (sixth in total), the camera remains completely static. This set is used to get a reference gyro and accelerometer series, used to determine the initial IMU orientation and estimate gyro and accelerometer variances. In the second of these sets (seventh in total), the camera is subjected to light shaking in the plane spanned by the viewing direction of the camera and the real-world gravity vector. The shaky checkerboard recording, called Q6318_checkerboard, is stabilized with the purely gyro-based stabilizer and the Kalman filter, using parameter estimates obtained from the first set.

3.1.2 Simulated data for sensor fusion

To be able to observe the sensor-fusion Kalman filter performance in a more controlled environment, a simulated data series was used to compare the Kalman filter position estimate to the position estimate one would get with a standard integrator. The simulator starts with an initial orientation for the IMU that is aligned with the global coordinate frame, represented by the quaternion

$$q = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{3.1}$$

At every time step t of the simulation, a small rotation ωdt is applied to the previous orientation quaternion, where ω is the simulated angular velocity, here given by

$$\omega(t) = \begin{bmatrix} 0 \\ 0 \\ \omega_z(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{\pi}{180}\sin(8\pi t) \end{bmatrix}. \tag{3.2}$$

The time step dt between each sample is 0.002 seconds, and the simulation simulates 5000 samples. Each simulated gyro sample is given by

$$\omega_{sim}(n) = \omega(ndt) + e + b \tag{3.3}$$

where $b = 0.1$ i and e is drawn from a normal distribution with zero mean and variance 0.001^2. The accelerometer measurements are simulated as

$$a_{sim}(n) = C(q_{ndt})g + e \tag{3.4}$$

where $C(q_{ndt})$ is the rotation matrix rotating the global frame to the IMU frame at time step ndt, g the simulated force of gravity, given as

$$g = \begin{bmatrix} 9.814 \\ 0 \\ 0 \end{bmatrix} \tag{3.5}$$

and e is drawn from a normal distribution with zero mean and variance 0.001^2. The position is then estimated with both the Kalman filter, as described in Section 3.2.2, and passing the samples through a high-pass filter and integrating the filtered gyroscope measurements according to Equation (2.44), which is similar to how the gyroscope-based stabilizer estimates position. The first 3000 simulated samples are used as a settling period for the filter and the orientation estimations are then compared for the final 2000 simulated samples.

3.1.3 Determining intrinsic camera parameters

Some of the methods require us to know the intrinsic camera matrix to be useful. We use the same approach as Forssén and Ringaby in [10], which is the approach described in the official OpenCV documentation [20]. OpenCV provides a camera calibration routine based on [21] that finds the intrinsic camera matrix given ten or more input images of a predefined calibration pattern with known world coordinates. We use 10 images of a 7x9 chessboard pattern taken with the camera settings used when recording the videos of the data sets. The chessboard has a known square size of 5x5mm.

3.2 Implementation

This section provides implementation details for the various stabilization approaches that have been implemented and evaluated.

3.2.1 Gyro stabilization

For the pure gyro stabilizer described in Section 2.2 we need to choose appropriate parameter values for the proportionality constant in equation(2.13), called the gain, the decay factor for the integrator θ, the decay β for the high pass filter in equation(2.14) and the capture time of each line in the sensor, t_l. The parameters depend on camera and zoom-level and can be found in table 3.2.1. The gain is found experimentally by comparing the pixel offsets from the gyro stabilizer to the pixel offsets obtained by the image-based global motion stabilizer on data sets that were not used for evaluation. The line time is based on known information about the image sensor.

Camera configuration	Gain	θ	β	t_l
P5655	0.015	0.98	0.998	$1.4 \cdot 10^{-8}$
Q6318 15x zoom	0.006	0.98	0.998	$0.7 \cdot 10^{-8}$
Q6318 31x zoom	0.012	0.98	0.998	$0.7 \cdot 10^{-8}$

Table 3.1: Parameters used for the pure gyro stabilizer

3.2.2 Gyroscope-Accelerometer sensor fusion

The Gyroscope-Accelerometer sensor fusion is implemented in accordance with the earlier derived scheme in section 2.3. The initial gyro bias estimate is set to 0. What remains is to get an initial estimate of the orientation, an estimate of the accelerometer bias (which is assumed to be constant) and finding reasonable settings for the matrices Q_d and R. To get the initial estimate of the orientation, the camera is left stationary for several seconds. The orientation is estimated by calculating the mean accelerometer measurement during that time, and using the orientation that would map g to the this mean acceleration as an initial estimate. The matrix Q_d represents the amount of uncertainty that is inserted into the system each time we propagate the nominal state by integrating new gyro measurements. This can be approximated as $\hat{\sigma_g}^2 \Delta t$, the variance of the gyro sensor multiplied by the time step Δt. The estimate of the gyro variance is obtained by calculating the variance of the gyro sequence from the stationary camera period. For the matrix R, representing the observation variance, we use the variance of the accelerometer series taken during the stationary period. We now need some way to use the orientation estimates to do image stabilization. If the Kalman filter estimates the two orientations $\hat{q}_t$ and $\hat{q}_{t+1}$ for two consecutive samples, one can calculate the small rotation ϕ required to rotate q_t to q_{t+1}. This small rotation ϕ can be accumulated to an x, y pixel displacement using Equation (2.13) and the same gain as the gyro stabilizer. The decay factor can be set to 1 since the orientation estimations in the Kalman filter should already be bias and error-corrected.

3.2.3 Image-based global motion stabilizer

OpenCV provides an implementation of the earlier described Good Features to Track algorithm and was used to get initial feature point estimates [20]. The number of feature points estimated is capped at a maximum of 200 features. Features are a minimum of 20 pixels apart, and the threshold for the minimial eigenvalue is set to 0.15. The features where tracked between frames using pyramidal Lukas-Kanade tracking, again using the available implementation in OpenCV [20]. The window size used was 21 x 21, and the number of pyramid levels three. As feature points are tracked between frames, they are gradually lost, partially due to uncertainties in the tracking, but also due to changes in the observed scene. Once the number of features have been reduced below 75% of the initial number of points, all points are re-estimated. A global motion vector between the consecutive frames are fitted using RANSAC. A set of corresponding points are randomly selected and used as the global motion vector. Points more than 2 pixels apart from the estimation are considered outliers. 50 RANSAC iterations were used.

3.2.4 Combined gyro and global motion stabilizer

An initial set of feature points are estimated using the OpenCV implementation of the feature point estimation algorithm described in Shi & Tomasi[16]. The number of feature points estimated is capped at 200. Features are a minimum of 20 pixels apart and the quality level is again set to 0.15. These features are tracked into the next frame using pyramidal Lucas-Kanade tracking described in Section 2.4.8, again using the OpenCV implementation of this algorithm. The window size is 7 x 7 and the number of pyramid levels is three. The gyro-based stabilization transform is applied to the tracked feature points, as described in Section 2.6.2, and a global motion vector between the gyro-corrected feature points from the previous frame. This is done in the same way as for the global motion stabilizer described in Section 3.2.3. When tracking the points, the inverse gyro-based stabilization transform is applied to the feature points from the previous frame to get a better initial guess, as described in 2.6.4. This stabilizer has an option to use downscaled versions of the input image for stabilization. The stabilizer has been tested at various input image resolutions, see results.

3.2.5 Background models

A background model can optionally be used with the combined gyro and global motion stabilizer to mask features in the foreground. The background model is updated every 10th frame, and a background mask is fetched from the model and used when doing feature point estimation and tracking. Points tracked in the foreground are excluded. The optical-flow based background model is used exclusively in the final stabilization algorithm.

3.2.6 Optical flow background model

Dense optical flow is estimated using the OpenCV implementation of Gunnar Farnebäcks method[20][22]. The median optical flow vector across the image is extracted, and the distance between the optical flow vector at each pixel and the median is calculated. The distance to the median is accumulated with a forgetting-factor of 0.6. If the distance to the median is greater than 5 between two consecutive frames, the pixel is considered to belong to the foreground. This, multiplied by the transient gain of the accumulator, yields a cutoff of 12.5 for the accumulated background image. The background model can optionally take downscaled versions of the image. The same cutoff for accumulated median-distance is used at all resolutions.

3.2.7 Image-based rolling shutter correction

In the rotation-spline based approach to image based rolling shutter correction, 3 rotation knots are used. The rotation knots are placed at the start of the first frame, at the midpoint between the two frames and at the end of the last frame. The number of blank lines between two images are determined by taking the difference between two consecutive capture times, subtracting the line time (same line times as used for the gyro stabilizer) parameters multiplied by the number of rows in the image, and dividing the result by the line time:

$$N_b = \frac{t_{f_{n+1}} - t_{f_n} - N_r t_{line}}{t_{line}} \tag{3.6}$$

Iterative least-squares optimization using the SciPy[23] implementation of the Levenberg-Marquadt algorithm (based on [24]) on (2.92) is used to fit the knot rotations. This is used with a tolerance of 0.1, meaning that the optimization is stopped when the cost function delta is smaller than 10 % of the current cost function value. To get the camera matrix K, we used the approach described in Section 3.1.3. The feature point estimations were obtained with the OpenCV implementation of [16] and tracking with the OpenCV implementation of the pyramidal Lucas-Kanade tracker. An extra outlier-rejection step was applied to the tracked points, same as used in the original article [10]. After the feature set x_k in I has been tracked into J, yielding y_k, the tracking is done in reverse: track y_k into I, forming $\tilde{x}_k$. If the pixel distance between a point x_k and $\tilde{x}_k$ is greater than 5, it is rejected as an outlier.

3.2.8 Camera implementation of tracking algorithms

Server-side feature point tracking can be done efficiently using the built-in methods in OpenCV. When running in camera systems certain OpenCV features are not available for use. Consequently, we provide our own implementation of CalcOpticalFlowPyrLK and GoodFeaturesToTrack using OpenCL, based on the descriptions of these algorithms in the earlier sections. The combined gyro and global motion stabilizer implementation is ported to the camera with the same parameters as described earlier, using our replacement functions instead of the OpenCV equivalent. The feature point tracking is done on GPU to leverage the parallel nature of the algorithm. This camera-side implementation is mainly used to benchmark performance of the feature point tracking.

3.3 Evaluation Procedure

Each stabilization algorithm described in the implementation section is used to stabilize the gathered footage. The algorithms are given one new frame at a time to stabilize, along with any gyroscope and accelerometer samples that have been gathered since the recording time of the previous frame. Each algorithm must then produce a stabilized version of the given frame, before the next one is provided. The stabilization is divided into two steps: the update step, where the stabilizer is fed a frame and IMU measurements and must update its internal state, and the stabilization step, where any stabilization transforms are applied to the image yielding the stabilized frame. Algorithm runtimes are measured on the update step, since the final image stabilization warp will need to be done by all stabilization algorithms anyway. The resulting stabilized video is then evaluated based on two different metrics: PSNR and structural similarity between consecutive frames. Information loss in the stabilized frame is also measured, to ensure algorithms do not crop the video excessively. Most importantly, the videos are inspected visually: it is important to remember that these metrics acts as proxies for the subjective experience of watching stabilized footage, which is the ultimate target for a stabilization algorithm. For the algorithms that utilize feature point tracking, a variety of metrics related to the performance of said metric is recorded: The number of feature points tracked and number of feature point re-estimations, number of outliers in RANSAC when fitting motion models. To evaluate the accuracy of the initial guess for the point locations with the gyro-corrected feature point tracking, the distance between the matched points in the new image and the initial guess is compared to the distance between the matched points and the point location in the previous image. Many of the stabilizers have stochastic elements in their implementations, so each set is stabilized five times with each stabilizer.

Chapter 4

Results

Table 4.1 contain stabilizer names used in the rest of the results sections, along with a short description.

Stabilizer	**Description**
No stabilizer	No stabilization is applied to the video.
Gyro	Purely gyro-scope based stabilization, implemented as in section 3.2.1
Kalman	Accelerometer + Gyroscope sensor fusion with a kalman filter, implemented as described in Section 3.2.2
KLT	The feature-point based global motion estimation stabilization algorithm, implemented as described in Section 3.2.3
Two pass Gyro + KLT	One pass of gyro stabilization, followed by a pass of KLT stabilization, as described in 2.6.1.
Hybrid 1920x1080	One pass-version of the gyro + KLT two pass stabilizer, implemented as described in Section 3.2.4. Input frames have dimensions 1920x1080.
Hybrid 1920x1080 BG	One pass-version of the gyro + KLT two pass stabilizer, implemented as described in Section 3.2.4. Input frames have dimensions 1920x1080. Uses an optical-flow based background model.
Hybrid 1280x653	One pass-version of the gyro + KLT two pass stabilizer, implemented as described in Section 3.2.4. Input frames are downscaled to dimensions 1280x653.
Hybrid 860x459	One pass-version of the gyro + KLT two pass stabilizer, implemented as described in Section 3.2.4. Input frames are downscaled to dimensions 860x459.
Hybrid 860x459 BG	One pass-version of the gyro + KLT two pass stabilizer, implemented as described in Section 3.2.4. Input frames are downscaled to dimensions 860x459. Uses an optical-flow based background model.

Table 4.1: Stabilizer names and descriptions.

4.1 Sensor fusion

4.1.1 Results on simulated data

Figure 4.1 contains the residuals and estimated angles from the Kalman filter and integrated high-pass filtered simulated gyroscope values. The simulated gyroscope and accelerometer variance was $[0.001^2, 0.001^2, 0.001^2]$ and the simulated gyroscope bias was $[0.001, 0.001, 0.001]$. For the Kalman filter, the mean residuals for the y and z angles was $1.475 \cdot 10^{-6}$ rad and $1.475 \cdot 10^{-6}$ rad respectively. The variance was $2.269 \cdot 10^{-9}$ rad^2 and $5.183 \cdot 10^{-9}$ rad^2 respectively. For the integrated high-pass filtered gyroscope measurements, the mean residuals for the estimated angle in y and z was $-3.183 \cdot 10^{-5}$ rad and $-4.587 \cdot 10^{-5}$ rad respectively. The variance was $5.508 \cdot 10^{-10}$ rad^2 and $8.744 \cdot 10^{-10}$ rad^2 respectively. A comparison with estimations from integrating the unfiltered gyroscope measurements can be found in figure 4.2.

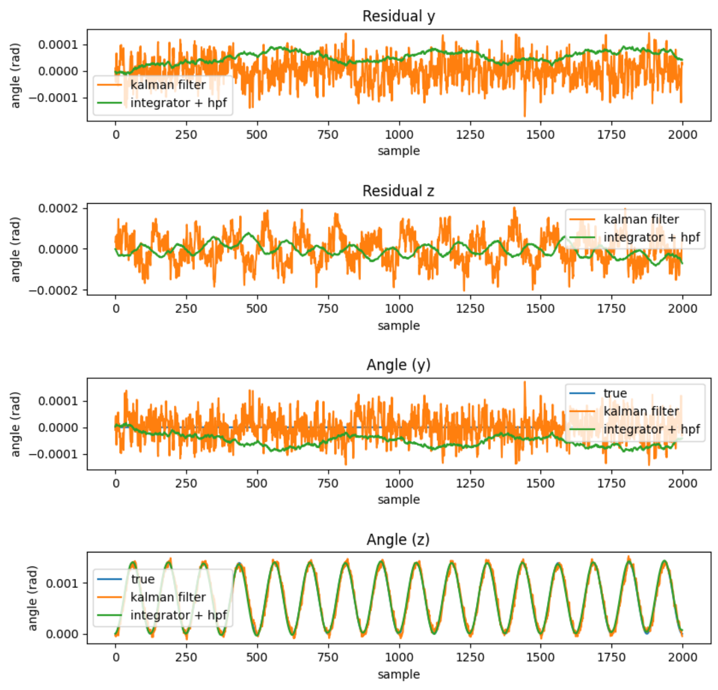

Figure 4.1: Estimated true orientation of the camera with the Kalman filter and integrated high-pass filtered gyro values on simulated data with equal gyroscope and accelerometer variances. The first plot shows the y-angle residuals, the second plot z-angle residuals, the third plot shows estimated y angle, the third plot shows estimated z angle

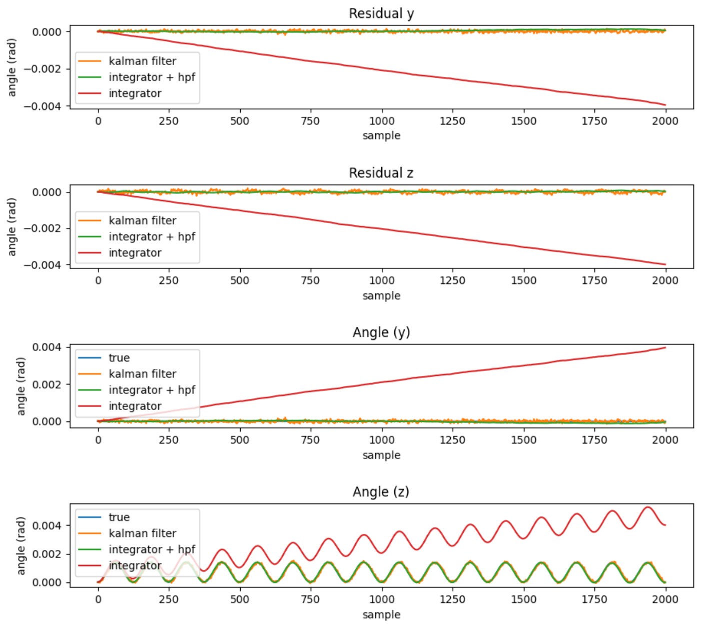

Figure 4.2: Estimated vs true angular offsets in y, z using the Kalman filter, integrated gyro values and integrated high-pass filtered gyro values from simulated data with equal gyroscope and accelerometer variances. The first plot shows the y-angle residuals, the second plot z-angle residuals, the third plot shows estimated y angle, the third plot shows estimated z angle

4.1.2 Results on real data

The standard deviation of the gyroscope measurements from a 45 second measurement series taken from the stationary Q6318 was
$\sigma_g = \begin{bmatrix}0.000763 & 0.000595 & 0.000829\end{bmatrix}$, and the estimated standard deviation for the accelerometer was $\sigma_a = \begin{bmatrix}0.0427 & 0.0431 & 0.0197\end{bmatrix}$. The mean accelerometer value for this series was $a_m = \begin{bmatrix}9.731 & -0.336 & 1.234\end{bmatrix}$. σ_g^2, σ_a^2 and a_m where used as the estimated gyroscope variance, accelerometer variance and to estimate the initial orientation in the Kalman filter respectively. The stabilization results using these parameters for the Kalman filter on the set Q6318_checkerboard can be found in table 4.1.2. The difference in image offsets per frame between the Kalman filter and the pure gyro stabilizer (using the same gain-parameter, see table 3.2.1) can be found in figure 4.3. To get a better feel for the expected Kalman filter performance on a data set of this kind, a simulation run using σ_g^2 and σ_a^2 as variance for gyroscope and accelerometer values respectively can be found in figure 4.4.

Stabilizer	Mean PSNR	average SSIM	Info loss
No stabilizer	333.84	0.982	0.002
Gyro	58.02	0.994	0.002
Kalman	52.22	0.923	0.005

Table 4.2: Stabilization metrics for pure gyro and sensor fusion Kalman filter on the Q6318_checkerboard data set

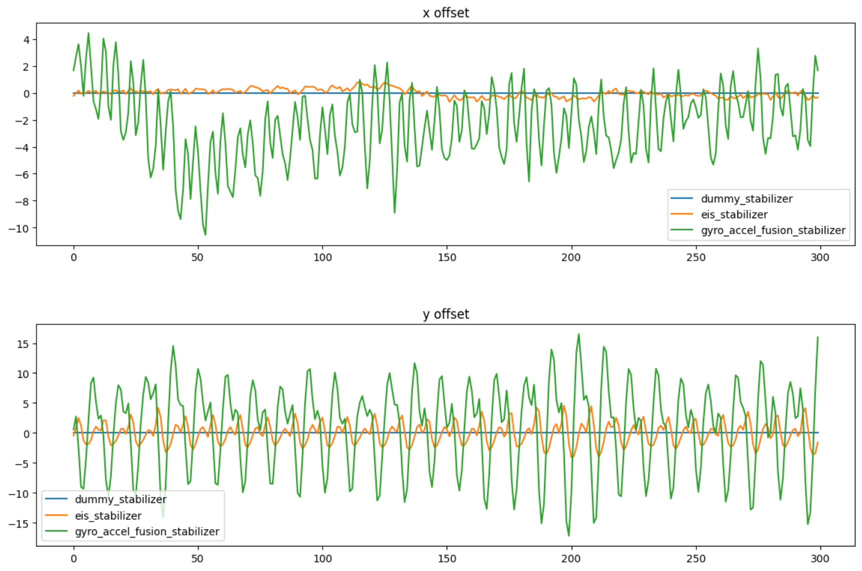

Figure 4.3: Pixel offsets per frame when stabilizing the Q6318_checkerboard data set for the pure gyro stabilizer and the sensor-fusion Kalman stabilizer

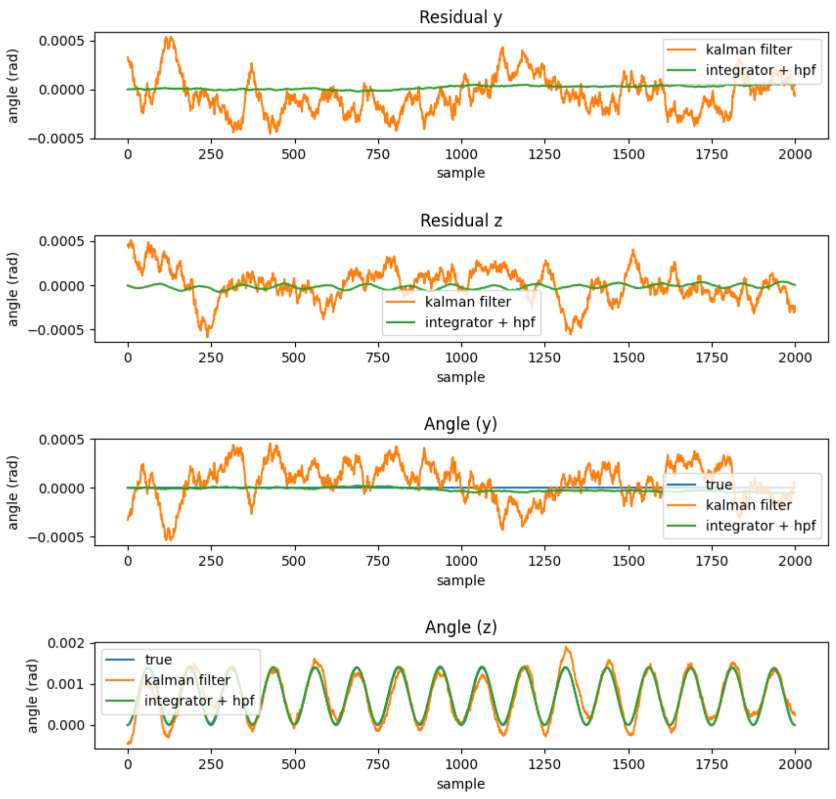

Figure 4.4: Estimated vs true angular offsets in y, z using the Kalman filter, integrated gyro values and integrated high-pass filtered gyro values from simulated data with the estimated variances for the gyroscope and the accelerometer from the Q6318_checkerboard set. The first plot shows the y-angle residuals, the second plot z-angle residuals, the third plot shows estimated y angle, the third plot shows estimated z angle

4.2 Stabilization results

The following section contains the qualitative stabilization results on footage from the P5655_road_zoom, Q6318_road_zoom_31, Q6318_building_zoom_31, Q6318_door_zoom_15 and Q6318_table_zoom_15 sets. Tables 4.2 through 4.2 contain mean PSNR and mean SSIM scores per frame for the various stabilizers that were evaluated on each respective data set, with error bars representing standard deviations taken over the five stabilization runs done on each data set. The reader is highly encouraged to look at the provided stabilized footage in the supplementary material to get a better idea of how the stabilizers actually perform.

Stabilizer	Mean PSNR	SSIM	Info loss
No stabilizer	15.124 ± 0.000	0.276 ± 0.000	$\mathbf{0.000 \pm 0.000}$
Gyro	20.483 ± 0.000	0.485 ± 0.000	0.016 ± 0.000
KLT	20.937 ± 0.032	0.548 ± 0.002	0.015 ± 0.000
Two Pass Gyro + KLT	$\mathbf{25.592} \pm 0.013$	$\mathbf{0.726} \pm 0.001$	0.022 ± 0.001
Hybrid 1920x1080	24.898 ± 0.006	0.695 ± 0.000	0.015 ± 0.000
Hybrid 1920x1080 BG	24.848 ± 0.019	0.693 ± 0.000	0.015 ± 0.000
Hybrid 1280x653	18.220 ± 0.016	0.379 ± 0.001	0.012 ± 0.000
Hybrid 860x459	16.478 ± 0.011	0.316 ± 0.001	0.014 ± 0.001
Hybrid 860x459 BG	16.484 ± 0.014	0.315 ± 0.001	0.011 ± 0.001

Table 4.3: Stabilization metrics for the P5655_Road_zoom data set. A high mean PSNR and high SSIM are indicative of good stabilization.

Stabilizer	Mean PSNR	SSIM	Info loss
No stabilizer	20.953 ± 0.000	0.566 ± 0.000	$\mathbf{0.000 \pm 0.000}$
Gyro	20.554 ± 0.000	0.545 ± 0.000	0.005 ± 0.000
KLT	$\mathbf{29.339} \pm 0.020$	$\mathbf{0.892} \pm 0.000$	0.006 ± 0.001
Two Pass Gyro + KLT	29.322 ± 0.042	0.887 ± 0.000	0.014 ± 0.001
Hybrid 1920x1080	28.679 ± 0.037	0.876 ± 0.001	0.009 ± 0.001
Hybrid 1920x1080 BG	28.732 ± 0.039	0.877 ± 0.001	0.010 ± 0.001
Hybrid 1280x653	25.650 ± 0.043	0.784 ± 0.001	0.006 ± 0.001
Hybrid 860x459	23.782 ± 0.043	0.701 ± 0.002	0.011 ± 0.003
Hybrid 860x459 BG	23.704 ± 0.028	0.697 ± 0.001	0.008 ± 0.002

Table 4.4: Stabilization metrics for the Q6318_road_zoom_31 data set. A high mean PSNR and high SSIM are indicative of good stabilization.

Stabilizer	Mean PSNR	SSIM	Info loss
No stabilizer	20.433 ± 0.000	0.636 ± 0.000	$\mathbf{0.000 \pm 0.000}$
Gyro	25.210 ± 0.000	0.822 ± 0.000	0.006 ± 0.000
KLT	28.404 ± 0.029	0.877 ± 0.001	0.007 ± 0.001
Two Pass Gyro + KLT	$\mathbf{29.872} \pm 0.037$	$\mathbf{0.905} \pm 0.000$	0.008 ± 0.000
Hybrid 1920x1080	29.121 ± 0.032	0.892 ± 0.000	0.006 ± 0.000
Hybrid 1920x1080 BG	29.238 ± 0.029	0.894 ± 0.000	0.006 ± 0.000
Hybrid 1280x653	25.112 ± 0.055	0.799 ± 0.001	0.007 ± 0.001
Hybrid 860x459	23.069 ± 0.044	0.730 ± 0.001	0.004 ± 0.000
Hybrid 860x459 BG	23.080 ± 0.032	0.731 ± 0.001	0.005 ± 0.001

Table 4.5: Stabilization metrics for the Q6318_building_zoom_31 data set. A high mean PSNR and high SSIM are indicative of good stabilization.

Stabilizer	Mean PSNR	SSIM	Info loss
No stabilizer	18.696 ± 0.000	0.549 ± 0.000	$\mathbf{0.000 \pm 0.000}$
Gyro	23.089 ± 0.000	0.746 ± 0.000	0.006 ± 0.000
KLT	26.758 ± 0.031	0.862 ± 0.001	0.012 ± 0.000
Two Pass Gyro + KLT	$\mathbf{28.638} \pm 0.031$	$\mathbf{0.902} \pm 0.000$	0.010 ± 0.000
Hybrid 1920x1080	27.852 ± 0.044	0.889 ± 0.001	0.010 ± 0.000
Hybrid 1920x1080 BG	28.150 ± 0.026	0.896 ± 0.000	0.009 ± 0.000
Hybrid 1280x653	23.660 ± 0.015	0.784 ± 0.001	0.010 ± 0.001
Hybrid 860x459	21.678 ± 0.031	0.697 ± 0.002	0.007 ± 0.002
Hybrid 860x459 BG	21.610 ± 0.071	0.693 ± 0.004	0.007 ± 0.002

Table 4.6: Stabilization metrics for the Q6318_table_zoom_15 set. A high mean PSNR and high SSIM are indicative of good stabilization.

Stabilizer	Mean PSNR	SSIM	Info loss
No stabilizer	20.955 ± 0.000	0.733 ± 0.000	$\mathbf{0.000 \pm 0.000}$
Gyro	22.265 ± 0.000	0.772 ± 0.000	0.004 ± 0.000
KLT	30.536 ± 0.031	0.911 ± 0.000	0.007 ± 0.000
Two Pass Gyro + KLT	31.080 ± 0.033	0.919 ± 0.000	0.008 ± 0.000
Hybrid 1920x1080	$\mathbf{31.172} \pm 0.052$	$\mathbf{0.927} \pm 0.001$	0.008 ± 0.001
Hybrid 1920x1080 BG	30.052 ± 1.217	0.912 ± 0.017	0.007 ± 0.001
Hybrid 1280x653	26.664 ± 0.038	0.863 ± 0.001	0.007 ± 0.001
Hybrid 860x459	24.315 ± 0.068	0.811 ± 0.002	0.005 ± 0.001
Hybrid 860x459 BG	24.223 ± 0.049	0.809 ± 0.002	0.006 ± 0.001

Table 4.7: Stabilization metrics for the Q6318_door_zoom_15 set

4.2.1 Metrics for the feature point tracking

This section contains various metrics illustrating the performance of the feature point tracking, particularly in the hybrid stabilizer. Various distance metrics related to the gyro-based initial guess for new feature point locations on the P5655_road_zoom set can be found in Table (4.8). In this table, x_est is the gyro-based guess, x the previous feature point locations, y_est the tracked feature point locations with x_est, y the tracked feature point locations with x. On the sets with good performance for the pure gyro stabilizer we expect $d(x_est, y_est)$ to be smaller than $d(x, y)$. The metrics are averaged across all runs and frames, with error bars representing the standard deviation across all measurements of that metric. Tables 4.9 to 4.13 contain metrics related to the tracking performance of the various hybrid stabilizers on the sets p5655_road_zoom, Q6318_road_zoom_31, Q6318_building_zoom_31, Q6318_table_zoom_15 and Q6318_door_zoom_15 sets. The measurements are averaged across the 5 stabilization runs on each set, with error bars representing the standard deviation for the average of each metric over the 5 runs. In these tables, track ratio is the number of initial estimated feature points to the number of current tracked ones, N_{re} the number of re-estimations done during a stabilization session, N_{masked} the average number of masked points per frame, $N_{tracked}$ the average number of tracked feature points per frame, $N_{inliers}$ the average number of RANSAC inliers per frame.

Test Case	$d(x_{est}, y_{est})$	$d(x, y)$	$d(y_{est}, y)$	$d(x, x_{est})$
p5655_road_zoom	8.942 ± 5.229	19.718 ± 12.580	3.711 ± 5.167	19.797 ± 12.809
q6318_road_zoom_31	5.625 ± 3.894	4.900 ± 2.884	0.255 ± 0.706	4.195 ± 2.711
q6318_building_zoom_31	4.044 ± 2.625	5.846 ± 3.467	0.163 ± 0.056	4.866 ± 3.080
q6318_table_zoom_15	7.146 ± 5.974	7.023 ± 4.503	0.286 ± 1.788	5.769 ± 4.027
q6318_door_zoom_15	2.956 ± 1.615	5.753 ± 3.117	0.179 ± 0.369	4.541 ± 2.662

Table 4.8: Statistics for the IMU-based initial feature point guess for the various data sets for the 1920x1080 hybrid stabilizer.

Stabilizer	track ratio	N_{re}	N_{masked}	$N_{tracked}$	$N_{inliers}$
Hybrid 1920x1080	0.833 ± 0.000	7.000 ± 0.000	0.000 ± 0.000	166.537 ± 0.000	128.901 ± 0.057
Hybrid 1920x1080 BG	0.811 ± 0.006	22.200 ± 2.638	1.242 ± 0.102	162.218 ± 1.267	135.965 ± 1.205
Hybrid 1280x653	0.809 ± 0.000	8.000 ± 0.000	0.000 ± 0.000	161.788 ± 0.000	134.508 ± 0.088
Hybrid 860x459	0.840 ± 0.000	6.000 ± 0.000	0.000 ± 0.000	160.978 ± 0.000	140.004 ± 0.072
Hybrid 860x459 BG	0.819 ± 0.002	25.600 ± 1.356	1.127 ± 0.061	152.893 ± 1.471	139.517 ± 1.404

Table 4.9: Hybrid stabilizer statistics for the P5655_road_zoom set.

Stabilizer	track ratio	N_{re}	N_{masked}	$N_{tracked}$	$N_{inliers}$
Hybrid 1920x1080	0.889 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	177.900 ± 0.000	158.069 ± 0.029
Hybrid 1920x1080 BG	0.826 ± 0.030	0.600 ± 0.490	0.064 ± 0.023	165.229 ± 6.095	150.404 ± 3.959
Hybrid 1280x653	0.900 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	179.970 ± 0.000	171.354 ± 0.018
Hybrid 860x459	0.889 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	163.543 ± 0.000	160.249 ± 0.010
Hybrid 860x459 BG	0.844 ± 0.002	0.000 ± 0.000	0.020 ± 0.001	155.212 ± 0.297	152.922 ± 0.276

Table 4.10: Hybrid stabilizer statistics for the Q6318_road_zoom_31 set

Stabilizer	track ratio	N_{re}	N_{masked}	$N_{tracked}$	$N_{inliers}$
Hybrid 1920x1080	0.900 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	150.302 ± 0.000	136.319 ± 0.031
Hybrid 1920x1080 BG	0.833 ± 0.005	0.000 ± 0.000	0.034 ± 0.004	139.108 ± 0.853	127.498 ± 0.585
Hybrid 1280x653	0.932 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	134.280 ± 0.000	128.219 ± 0.048
Hybrid 860x459	0.948 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	117.602 ± 0.000	115.759 ± 0.026
Hybrid 860x459 BG	0.944 ± 0.004	0.000 ± 0.000	0.001 ± 0.001	117.003 ± 0.556	115.170 ± 0.527

Table 4.11: Hybrid stabilizer statistics for the Q6318_building_zoom_31 set

Stabilizer	track ratio	N_{re}	N_{masked}	$N_{tracked}$	$N_{inliers}$
Hybrid 1920x1080	0.816 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	163.298 ± 0.000	146.084 ± 0.030
Hybrid 1920x1080 BG	0.889 ± 0.007	1.000 ± 0.000	0.085 ± 0.003	177.763 ± 1.351	161.453 ± 1.181
Hybrid 1280x653	0.863 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	172.602 ± 0.000	164.095 ± 0.013
Hybrid 860x459	0.854 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	150.385 ± 0.000	148.393 ± 0.010
Hybrid 860x459 BG	0.815 ± 0.008	0.000 ± 0.000	0.025 ± 0.005	143.423 ± 1.356	142.067 ± 1.286

Table 4.12: Hybrid stabilizer statistics for the Q6318_table_zoom_15 set

Stabilizer	track ratio	N_{re}	N_{masked}	$N_{tracked}$	$N_{inliers}$
Hybrid 1920x1080	0.857 ± 0.000	1.000 ± 0.000	0.000 ± 0.000	171.438 ± 0.000	145.172 ± 0.025
Hybrid 1920x1080 BG	0.835 ± 0.021	8.400 ± 4.030	0.567 ± 0.301	161.147 ± 10.450	146.135 ± 9.135
Hybrid 1280x653	0.911 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	174.075 ± 0.000	160.615 ± 0.032
Hybrid 860x459	0.917 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	114.628 ± 0.000	111.685 ± 0.022
Hybrid 860x459 BG	0.912 ± 0.003	0.000 ± 0.000	0.001 ± 0.001	113.994 ± 0.317	111.184 ± 0.242

Table 4.13: Hybrid stabilizer statistics for the Q6318_door_zoom_15 set

4.2.2 Running times

Table 4.14 contains the average update time across all frames and test cases for the host-side implementations of the Gyro, KLT, Two-pass Gyro + KLT, 1920x1080 hybrid and the 860x439 hybrid stabilizers. Update time refers the amount of time it takes to update the internal state of the algorithm, disregarding the time it takes to use that information to apply the stabilization transform to the frame . Error bars are the standard deviations of the update times. The figures 4.5, 4.6, 4.7 show logarithm scale histogram of the update times for the Gyro, KLT and 1920x1080 Hybrid stabilizers. The on-camera implementation of the feature point tracking algorithm in section 2.4.9 with three pyramid levels, 200 feature points, 9x9 integration window and five iterations had an average running time of 0.137 seconds per frame.

Stabilizer	Update time (s)
Gyro	0.003 ± 0.001
KLT	0.013 ± 0.002
Two pass Gyro + KLT	0.130 ± 0.009
Hybrid 1920x1080	0.033 ± 0.003
Hybrid 860x439	0.027 ± 0.003

Table 4.14: Average update time across all test cases and runs for some select stabilizers

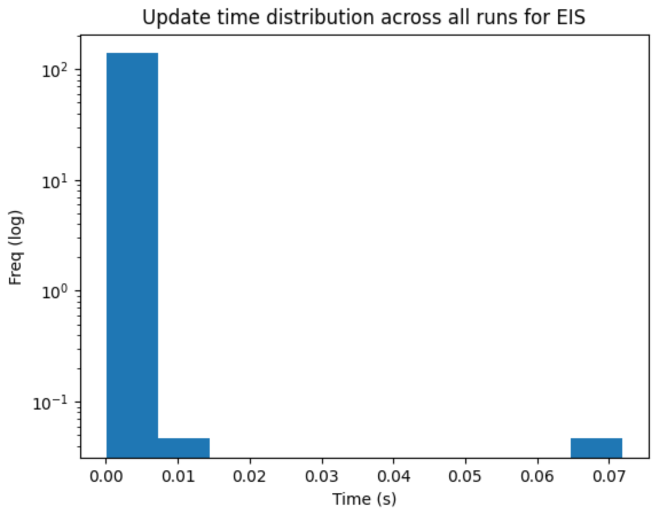

Figure 4.5: Distribution of update times for Gyro

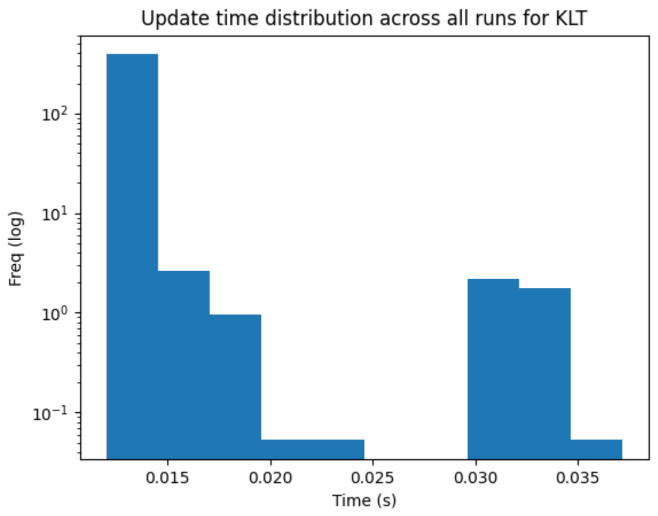

Figure 4.6: Distribution of update times for the KLT stabilizer

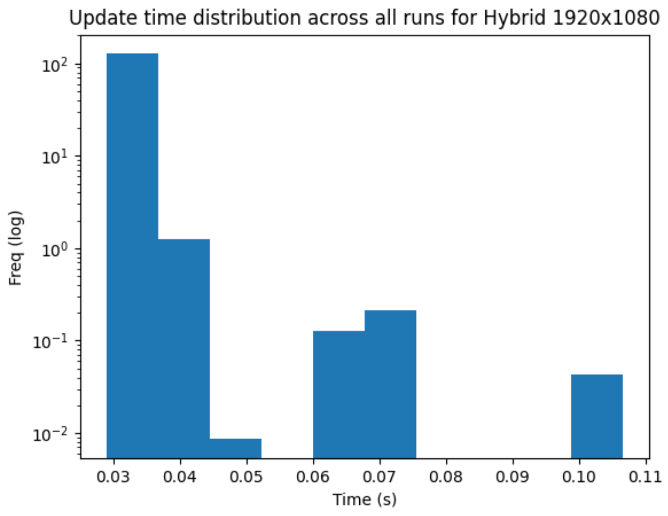

Figure 4.7: Distribution of update times for the hybrid stabilizer

4.3 Image-based rolling shutter correction

Our implementation of the rolling-shutter correction algorithm presented by Forssén and Ringaby was evaluated against their provided artificial data set.

The average structural similarity between corrected frames and ground truth frames in the house_rot_1_B40 set was 0.7547, compared to the average structural similarity of 0.4884 between ground truth frames and uncorrected frames. To get a better feel for the result, see figure 4.8 and 4.9 for reference. The algorithm was also tested on a 60 frame subset of the P5655_road_zoom video. The mean PSNR between consecutive frames was 16.418 and the mean structural similarity 0.3301, compared to a mean PSNR of 16.412 and a mean structural similarity of 0.3266 for consecutive uncorrected frames. The results were too inconsistent to be used in a stabilization algorithm, since some frames were left heavily distorted compared to the original. This is best illustrated by the example provided in figure 4.10.

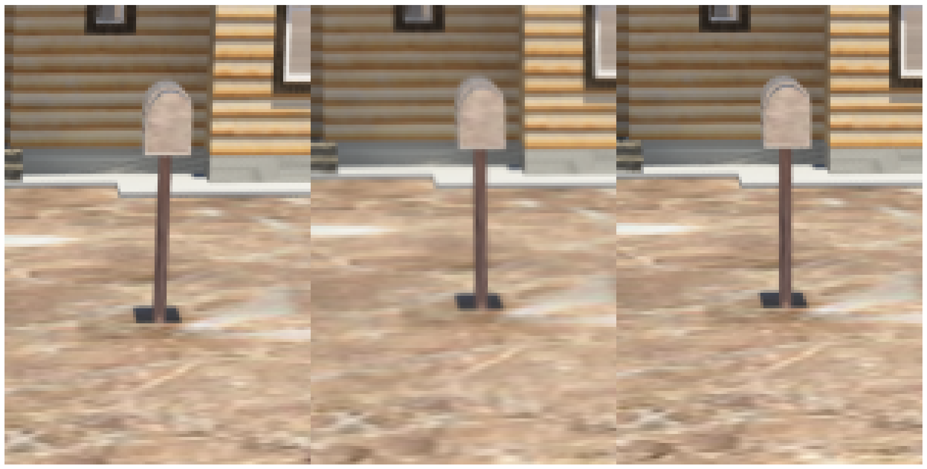

Figure 4.8: Side-by-side images of uncorrected image, corrected image with our implementation and ground truth from the data set provided by Forssén and Ringaby

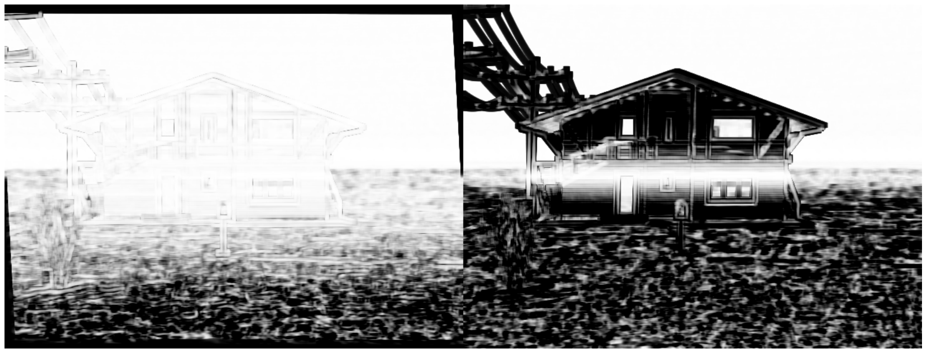

Figure 4.9: Image difference between stabilized image and ground truth and unstabilized image and ground truth

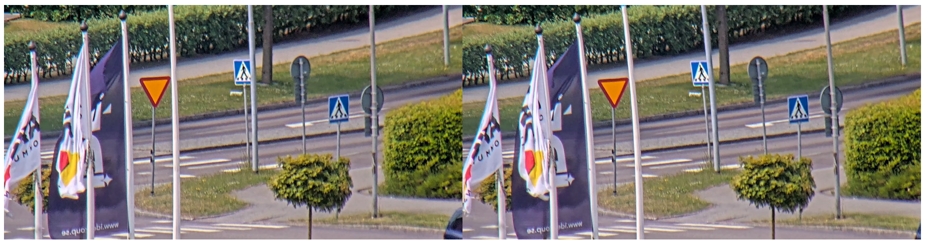

Figure 4.10: Two consecutive corrected frames from the road_zoom data set. The flag poles in the right image are heavily bent.

Chapter 5

Discussion

5.1 Sensor fusion viability

The possibility to use a sensor-fusion between the gyroscope and accelerometer was initially considered a promising approach to the problem. A solution like this is attractive for many reasons. It would be easy to fit into the current stabilization framework and have little performance overhead. Unfortunately, the sensor fusion approach investigated in this thesis does not seem to be viable. The stabilization result in table 4.1.2 is worse than not applying any stabilization at all. Worth noting here is that the high mean PSNR for the unstabilized footage is likely due to how the video is compressed on export with a very white background, but the structural similarity score still suggests the same thing. In hindsight, this is perhaps not too unexpected. The accelerometer is mainly used to correct accumulating errors due to biases in the gyroscope that the accelerometer can observe due to the direct estimation of the camera orientation. In the surveillance use case the position of the camera is not supposed to change, meaning any drift from zero mean is known to be an error and can safely be filtered out. Use of the Kalman filter could still be warranted however, since it allows us to combine two different sources of information, gaining more data and hopefully a more accurate estimation of the camera position. Unfortunately, the accelerometer is too noisy for this to happen, having an estimated variance that is roughly 2 orders of magnitude larger than the gyroscope. When simulating with similar variance on gyroscope and accelerometer (see figure 4.1), the Kalman filter performs better comparatively to the high-pass filtered integrator and when simulating with the real estimated gyroscope and accelerometer variances, the Kalman filter position estimate is way noisier than that of the integrator, while the high pass filter effectively stops the gyroscope bias from causing drift in the integrator (see figure 4.4 and compare with the drift for the unfiltered integrator in figure 4.2). A Kalman filter approach would probably be more viable if the camera was not statically mounted. In those cases, there can be camera movement that is desired, which is more difficult to handle with a pure gyro-based stabilization algorithm.

5.2 Stabilization results

5.2.1 General observations

On the P5655_road_zoom data set, the gyro + KLT hybrid stabilizer performed better than either one did individually, while both the gyro and KLT stabilizer offered significant image quality improvements than the unstabilized video. A visual inspection of the results suggest that the gyro stabilizer does not properly align the frames, whereas it reduces rolling shutter wobble in the image significantly, while the opposite is true for the KLT stabilizer, which has aligned images but significant rolling shutter artifacts. The result was similar for the set Q6318_building_zoom_31 and Q6318_table_zoom_15: Both gyro and KLT stabilization offers improvements over the unstabilized input video and combining the two yields the best result. For the sets Q6318_road_zoom_31 and Q6318_table_zoom_15 the gyro stabilizer did not offer any real improvement compared to the unstabilized videos. The KLT stabilizer reduced the jitter in the video significantly, but still suffers from significant rolling shutter artifacts. The combined approach performed somewhat worse than the pure KLT, which is perhaps not too unexpected when the gyro stabilizer fails to stabilize, since the initial feature point location estimate will not be correct. In general, the one-pass Hybrid stabilizer and the gyro + KLT two pass version had similar qualitative performance.

It is important to mention that the fact that the stabilization has been done server-side on exported data can impact the result compared to a hypothetical camera-side solution. When the video is saved and exported, the video frames will have gone through several post-processing steps such as denoising and sharpening. If an image-based stabilization was implemented direclty on the camera it might gain access to the frames for analysis before these steps are applied, affecting for example feature point tracking accuracy.

5.2.2 Downscaled versions of the hybrid stabilizer

In general, the hybrid stabilizer performed worse and worse the lower the resolution of the video frame used for input to the stabilizer was, which is expected: this is trading off accuracy for performance. What is noteworthy is that for the test sets where the gyro stabilizer performs well, downscaling the hybrid stabilizer to even 1280x653 meant losing any gain to using the pure gyro stabilizer. This would suggest that if a hybrid solution is to be used, it should most likely be operating on the input frame resolution.

5.2.3 Running times

When looking at the average stabilizer update times on the server side implementations in table 4.14, the computational efficiency of the gyro stabilizer is evident, which is in line with our initial expectations. Using the hybrid stabilizer approach instead of relying purely on feature point tracking was roughly three times slower, but approximately four times faster than doing two stabilization passes, with equal qualitative results. The actual speed-up factor will be different in an on-camera implementation, since the application of the gyro stabilization step will be implemented more efficiently, but the two algorithms perform roughly the same computational steps, with the exception of one not having to apply the gyro stabilization to the full frame, so a significant gain can still be expected. Downscaling the input image did improve the update time for the hybrid stabilizer somewhat, but since the qualitative results generally were not particularly impressive for the downscaled hybrid stabilizer, it is unlikely to be used anyway. When looking at figures 4.5, 4.6 and 4.7 it is clear that the image based stabilizers here have more outliers where update times are significantly slower than the average. This is due to the occasional re-estimation of feature points, which is more expensive than the feature point tracking. One could work around this by implementing a solution where the feature point re-estimation is non-blocking. When features start running low, a re-estimation is triggered in a separate process. When the re-estimation is finished, a few frames will have passed. The re-estimated points are tracked to the current frame and then swapped with the current estimates, removing the outlier cases where updates are significantly slower than average performance.

The main bottleneck for an image-based stabilizer to be viable on target is the requirement on real time feature point tracking. It is the most expensive computational step in the KLT and Hybrid stabilizers and an integral part of many image-based stabilization methods. With the current GPU implementation on target, the performance is not good enough to be considered real time when tracking points in full resolution. An average tracking time of 0.137 seconds per frame corresponds to around 7 FPS, which is quite far from the 25-30 FPS the cameras record in. Performance might improve somewhat with a more optimized implementation, but it is likely that stronger hardware would be required to reach acceptable performance.

5.2.4 Gyro-based initial guess for KLT stabilization

For the test sets P5655_road_zoom, Q6318_table_zoom_15 and Q6318_building_zoom_31 the gyro stabilizer performed well (as discussed earlier) and applying inverse gyro stabilization to the feature points to produce a new initial guess placed the initial feature point guess significantly closer to the final tracked locations compared to when not using the guess. In the remaining cases, where the gyro stabilizer performed poorly, the initial guess is about as far away from the final point as just using the previous feature point locations, which is expected.

5.2.5 Background model performance

The background models used with the hybrid stabilizer did not offer any significant performance differences to the version without it when looking at the image stabilization metrics. When looking at the tracking performance, one would look for something like higher inlier counts in RANSAC compared to the version without it, but this does not seem to be the case. While the background model did not hurt the qualitative results, it does not seem to offer much improvement over just using RANSAC for model fitting. Having a good outlier rejection criteria in RANSAC is significantly cheaper computationally as well. That being said, it is possible that a background model could still increase stabilizer robustness if more time is spent tuning the background model parameters.

5.3 Image based rolling shutter correction

The image based rolling shutter correction approaches used by Forssén and Ringaby yielded disappointing results on our data sets. There could be errors in the implementation since both the authors do not provide official source code for their algorithms to compare with. When comparing our implementations of their algorithms to other third party implementations and evaluating on the reference set provided by Forssén and Ringaby there does not seem to be notable differences. One possible explanation for this is a high requirement on feature points spread across the entirety of the image, as discussed in for instance Grundmann et. al [11]. They make modifications to the feature point detection algorithm to find more features in less distinguishable areas of the image. Investigating if these modifications would yield better results for our use-case would be an interesting point of further research. It is worth noting however that more feature points comes at a notable performance cost in an already limited hardware environment and it might not be worth adding that correction layer if this cost is too high. It is also worth noting that the algorithm used to fit the rotation spline is a least-squares algorithm instead of RANSAC, making the solution more susceptible to outliers. We used the outlier rejection method suggested by Forssén and Ringaby of doing reverse tracking, but it is possible that more could be done to remove potential outliers from the set. Even if the algorithm would produce qualitatively good results, the least squares minimization process required to fit the quite large motion model took significant time to execute even server-side, where performance is expected to be significantly better than on camera. This could probably be accelerated with dedicated hardware, but presents a very severe limitation regardless.

5.4 Conclusions & Further work

The hybrid stabilization approach showed some qualitative promise, while having the very desirable property of operating on top of the gyro-based stabilization layer. The main bottleneck is attaining sufficient feature point tracking speeds on camera, which seems out of reach at full resolution given current hardware, but it is possible that a more efficient implementation would enable this in the future. Introducing scene dependence by using image analysis remains a major drawback. Including the accelerometer does not seem to improve image stabilization, in large part due to the noise levels of the sensor and the fact that the accumulation of errors can be addressed with the assumption of all camera motion being unwanted. To address the proposed research question: Adding a simple image-based stabilization layer to a gyroscope-based stabilizer improves the qualitative performance significantly, but real-time performance is barely out of reach with current hardware. One interesting approach to study further would be to expand the Kalman filter to include motion estimates from the feature points, similar to [6] and [17]. The corresponding point pairs likely produce more accurate estimates of the camera orientation than the accelerometer. The main challenge here is probably to find a good model for rolling shutter and how it influences the credibility of the feature point estimates (both [6] and [17] assume global shutter). The image-based rolling shutter correction was largely unsuccessful. Even with better qualitative results, the computational requirements of the approach used would remain daunting. Further exploring real-time rolling shutter correction using image analysis could improve results greatly.

Bibliography

[1] Carl Olsson. *Lecture Notes in Computer Vision.* Feb. 2022.

[2] Wikimedia Commons. *Pinhole Camera.* 2019. URL: `https://upload.wikimedia.org/wikipedia/commons/thumb/3/3b/Pinhole-camera.svg/1280px-Pinhole-camera.svg.png`.

[3] Jongmin Baek Alexandre Karpenko David Jacobs and Marc Levoy. *Digital Video Stabilization and Rolling Shutter Correction using Gyroscopes.* Tech. rep. CSTR 2011-03. Department of Computer Science, Stanford University, 2011.

[4] Nikolas Trawny and Stergios I. Roumeliotis. *Digital Video Stabilization and Rolling Shutter Correction using Gyroscopes.* Tech. rep. 2005-02. Department of Computer Science Engineering, University of Minnesota, 2005.

[5] Joan Solà. "Quaternion kinematics for the error-state Kalman filter". In: *CoRR* abs/1711.02508 (2017). arXiv: `1711.02508`. URL: `http://arxiv.org/abs/1711.02508`.

[6] Faraz M. Mirzaei and Stergios I. Roumeliotis. "A Kalman Filter-Based Algorithm for IMU-Camera Calibration: Observability Analysis and Performance Evaluation". In: *IEEE Transactions on Robotics* 24.5 (Oct. 2008), pp. 1143–1156. ISSN: 1941-0468. DOI: `10.1109/TRO.2008.2004486`.

[7] Andreas Jakobsson. "An introduction to time series modeling /". In: Fourth edition. Lund : Studentlitteratur, [2021]. Chap. 8.5.

[8] M. Grundmann, V. Kwatra, and I. Essa. "Auto-directed video stabilization with robust L1 optimal camera paths." In: *CVPR 2011, Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on* (2011), pp. 225–232. ISSN: 978-1-4577-0394-2.

[9] Feng Liu, Michael Gleicher, Hailin Jin, and Aseem Agarwala. "Content-Preserving Warps for 3D Video Stabilization". In: *ACM Trans. Graph.* 28.3 (July 2009). ISSN: 0730-0301. DOI: `10.1145/1531326.1531350`.

[10] PE. Forssén and E. Ringaby. "Efficient Video Rectification and Stabilisation for Cell-Phones." In: *International Journal of Computer Vision* 96.3 (2012), pp. 335–352. ISSN: 0920-5691.

[11] M. Grundmann, V. Kwatra, D. Castro, and I. Essa. "Calibration-free rolling shutter removal." In: *2012 IEEE International Conference on Computational Photography (ICCP), Computational Photography (ICCP), 2012 IEEE International Conference on* (2012), pp. 1–8. ISSN: 978-1-4673-1660-6.

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

[13] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[14] Jean-Yves Bouguet. "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm". In: *Intel corporation* 5.1-10 (2001), p. 4.

[15] Bruce D Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *IJCAI'81: 7th international joint conference on Artificial intelligence*. Vol. 2. Vancouver, Canada, Aug. 1981, pp. 674–679. URL: https://hal.science/hal-03697340.

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

[17] Michael J. Smith, Alexander Boxerbaum, Gilbert L. Peterson, and Roger D. Quinn. "Electronic image stabilization using optical flow with inertial fusion". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 1146–1153. DOI: 10.1109/IROS.2010.5651113.

[18] Zhenmei Shi, Fuhao Shi, Wei-Sheng Lai, Chia-Kai Liang, and Yingyu Liang. "Deep Online Fused Video Stabilization". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2022, pp. 1250–1258.

[19] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[20] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[21] Z. Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718.

[22] Gunnar Farnebäck. “Two-frame motion estimation based on polynomial expansion”. In: *Image Analysis: 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden, June 29–July 2, 2003 Proceedings 13*. Springer. 2003, pp. 363–370.

[23] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: `10.1038/s41592-019-0686-2`.

[24] Jorge J. Moré. “The Levenberg-Marquardt algorithm: Implementation and theory”. In: *Numerical Analysis*. Ed. by G. A. Watson. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 105–116. ISBN: 978-3-540-35972-2.