FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

Odometry from RGB-D Cameras for Autonomous Quadrocopters

Christian Kerl

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

Odometry from RGB-D Cameras for Autonomous Quadrocopters

Odometrie aus RGB-D Kameras für Autonome Quadrokopter

Author:	Christian Kerl
Supervisor:	Prof. Dr. Daniel Cremers
Advisor:	Dr. Jürgen Sturm
Submission Date:	November 12, 2012

I assure the single handed composition of this master's thesis only supported by declared resources.

Ich versichere, dass ich diese Master's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Munich, November 12, 2012 Christian Kerl

Acknowledgments

First of all, I would like to thank Prof. Dr. Daniel Cremers for offering me the possibility to pursue my master's thesis at his chair. Second, I thank my advisor Dr. Jürgen Sturm for his continuous support and many fruitful discussions. Thanks to Tayyab Naseer for his assistance during the flight experiments. Further, I thank Markus Achtelik and Stephan Weiss for providing direct support with the asctec_mav_framework and ethzasl_sensor_fusion packages. Special thanks to Rachel Hornung for proof reading this thesis and her valuable comments.

Abstract

This master's thesis presents a robust, dense visual odometry method applicable to the stabilization of quadrocopters. In contrast to previous approaches using only sparse visual feature points in the image, the motion of the camera is estimated by aligning consecutive images based on the photo-consistency assumption using all image information. A novel RGB-D camera providing color and depth information is used. Robustness is achieved by embedding the motion estimation in a Bayesian framework. The involved probability distributions are modeled based on empirical data. In extensive experiments on synthetic and real datasets the superior performance in terms of accuracy and speed in comparison to previous implementations is validated. Furthermore, experiments on a quadrocopter demonstrate the suitability of the approach for stable motion estimation in realtime control loops.

Contents

1. Introduction

In recent years unmanned aerial vehicles (UAVs) have increasingly been used in several application areas like surveillance, mapping, disaster monitoring, industrial inspection and aerial photography.

Quadrocopters are one kind of UAVs, which are capable of flying at low speed and to hover. Additionally they offer high manoeuvrability making them ideally suited for flying in space restricted environments like buildings. Recent examples of UAVs being deployed to inspect damaged buildings are the following: A quadrocopter with cameras is used to explore a damaged church in northern Italy [1]. Figure 1.1 shows the quadrocopter during flight in the church. Similarly, a consumer grade Parrot AR.Drone has been deployed to inspect a cathedral in New Zealand, which was damaged during an earthquake [20]. A third example is the inspection of the destroyed nuclear reactors in Fukushima, Japan by an UAV [18]. In all these examples the quadrocopters have been remotely controlled by human pilots. Remote control requires a stable connection to transmit the control commands and the video streams from on-board cameras. Alternatively to video streams, the operator has to maintain a line of sight to the quadrocopter. These requirements limit the applicability of such systems. Therefore, quadrocopters which autonomously hold position and avoid obstacles providing shared autonomy to the pilot would be a great enhancement. Then, the pilots main task is to select new positions to get the best possible view of the situation.

Autonomous quadrocopters are challenging technical systems, because they are inherently unstable due to the fact that they are only flying with the lift created by their rotors. For stable flight quadrocopters require controllers running at high update rates adapting the speed of the different rotors to hold position or fly at a given speed. The need for fast controllers imposes the requirement to run most of the related computations on-board. In contrast, due to their size and payload restrictions only a limited amount of computational resources can be carried on-board.

For precise control a measurement of the position relative to a fixed coordinate system is required. In outdoor applications the Global Positioning System (GPS) or similar satellite-

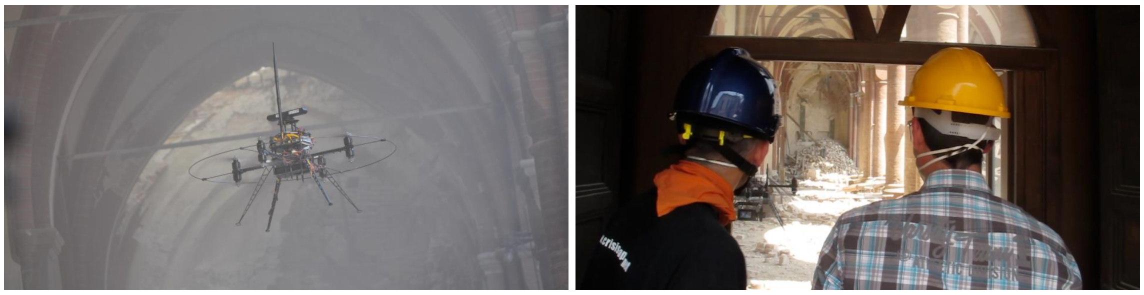


Figure 1.1.: Quadrocopter with RGB-D camera inspecting a church damaged during an earthquake (left). The quadrocopter is entirely controlled by human operators (right). [2].

based systems can be used. When flying in buildings or other GPS-denied areas different sensors have to be used for position estimation.

In case the robot carries all the sensors used for this position estimation, it is called odometry. One possible type of sensor are cameras. The ego motion can be estimated from the camera image stream. This odometry method is named visual odometry. Several systems exist either using monocular or stereo cameras. The problem simplifies with a stereo camera, because the absolute depth of the scene is known. The standard approaches reduce the amount of data to process by extracting feature points from the images. Afterwards, the motion is estimated from feature correspondences between two or more images. Those approaches are called sparse visual odometry. Recently, with the advent of more powerful computing resources, several dense visual odometry methods have been proposed. Another enabling technology are novel commodity stereo cameras providing accurate color and depth information at relatively high resolutions. The cameras are known as RGB-D cameras. The dense methods have the advantage that they are more accurate and that the processing pipeline is simpler. While sparse visual odometry methods have already been used to stabilize and control quadrocopter, this has not been done using dense approaches.

1.1. Research Question

The main research question assessed in this master's thesis is, whether it is possible to implement a dense visual odometry approach using images from RGB-D cameras to stabilize and control the position of a quadrocopter. The main challenge is to make the method robust enough to be used in real world environments, i.e. provide accurate estimates over a long time. A second requirement is to optimize the method so it can run in realtime on-board a quadrocopter.

1.2. Outline

Chapter 2 introduces the hardware platform and basic theoretical concepts used in this thesis. In particular, the employed quadrocopter platform and RGB-D camera are described. Furthermore, the models for the image formation process in the camera and for the motion representation are detailed. The chapter closes with an introduction to the least squares technique used for parameter estimation. The following chapter 3 gives an overview of existing sparse and dense visual odometry systems and their limitations. Chapter 4 describes the approach developed in this thesis. Afterwards, chapter 5 presents the results of an extensive set of experiments assessing the performance on different datasets and during flight on the quadrocopter. A third group of experiments is carried out to determine the influence of different parameters and their optimal values. Finally, chapter 6 summarizes the achievements and provides an outlook on possible extensions and future research.

2. Background

This chapter introduces the main hardware components. Further, the theoretic concepts applied in this thesis are explained.

2.1. Quadrocopter

A small, unmanned quadrocopter is the main hardware platform used in the course of this thesis. Quadrocopters are an emerging platform used for example for surveillance, inspection, mapping and aerial photography. This section introduces the basic concepts of quadrocopters and describes the employed Ascending Technologies Pelican quadrocopter in detail.

2.1.1. General

Quadrocopter are rotary wing aircraft with four rotors [59]. The speed of the rotors solely controls the lift and attitude. This is in contrast to helicopters, which change the configuration of the blades of the main rotor through a mechanical construction. Therefore, a quadrocopter has less moving parts, which simplifies its construction and maintenance. In contrast, the precise control of the rotation speed of the rotors, c.q. stabilization is more difficult. As all rotor aircraft quadrocopter can take off vertically, hover and fly at low speeds.

Figure 2.1 depicts the setup of the rotors and a convenient option to choose the local coordinate system, albeit the coordinate frame can be attached arbitrarily. Two opposite rotors form a pair rotating in the same direction. The other pair rotates in the opposing direction. To change height the speed of all four rotors is increased simultaneously or decreased. Rotation around the x axis (roll) is achieved by reducing the speed of the left rotor and increasing the speed of the right one, or vice versa. Similarly, different rotation speeds of the front and rear rotor cause a rotation around the y axis (pitch). Differences in the speed of the rotor pairs cause a rotation around the z axis (yaw).

2.1.2. Ascending Technologies Pelican

The Ascending Technologies (AscTec) Pelican is a medium sized quadrocopter with a width and length of 72 cm and a height of 26 cm. Therefore, it is still small enough to fly in indoor environments, but is able to carry a payload of 650 g [17]. Figure 2.2 shows a side-view with all payload, which is required for the approach and experiments, mounted. The Pelican is equipped with two processor boards, an inertial measurement unit (IMU), a monocular grayscale camera, and an ASUS Xtion PRO LIVE RGB-D camera (see section 2.2).

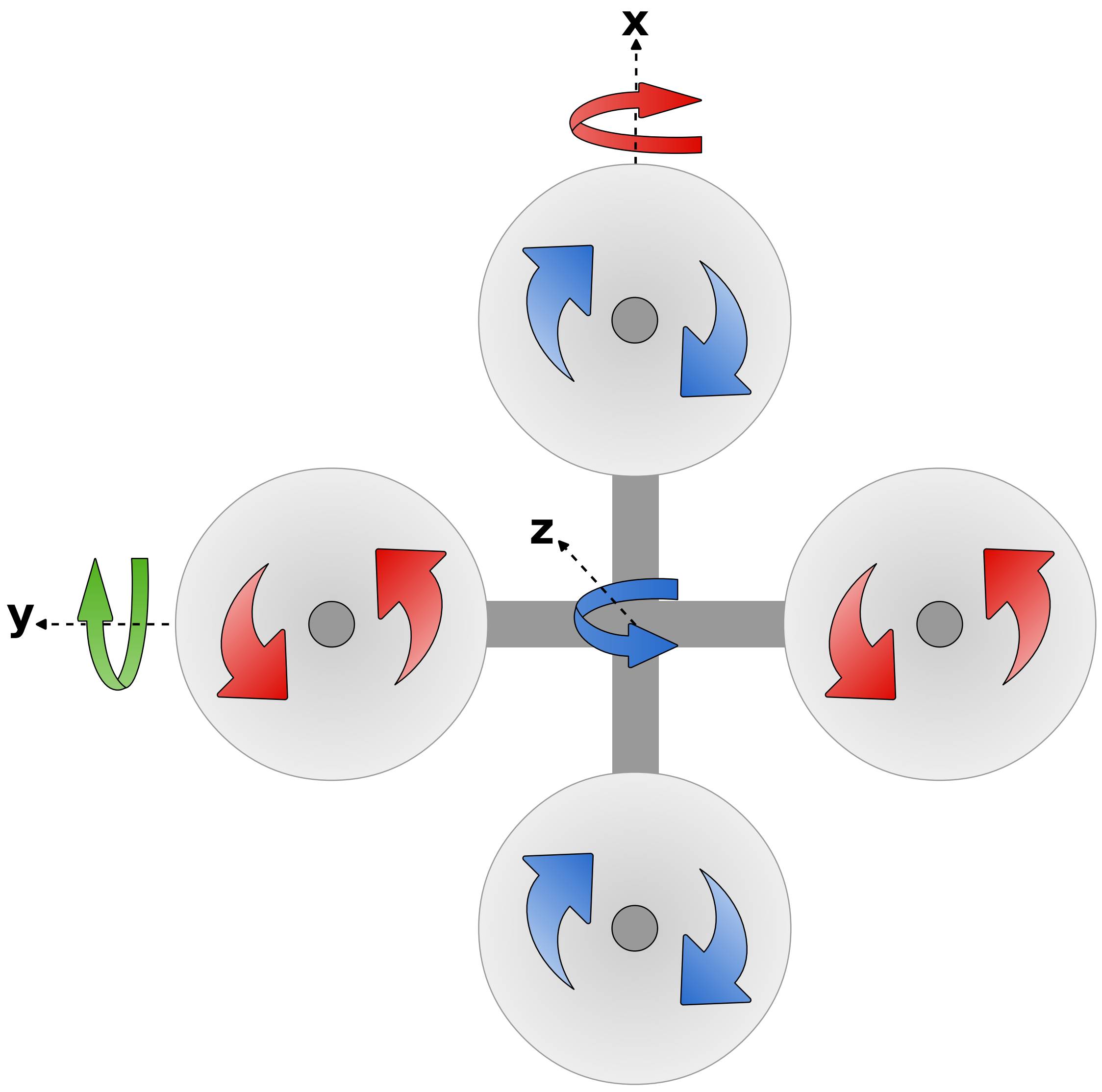


Figure 2.1.: Rotor setup of a quadrocopter and exemplary coordinate system. Two rotors form a pair rotating in the same direction. The other pair rotates in the opposite direction. By adjusting the rotation speed of the rotors the quadrocopter can change height, rotate around the x axis (roll), y axis (pitch), and z axis (yaw).

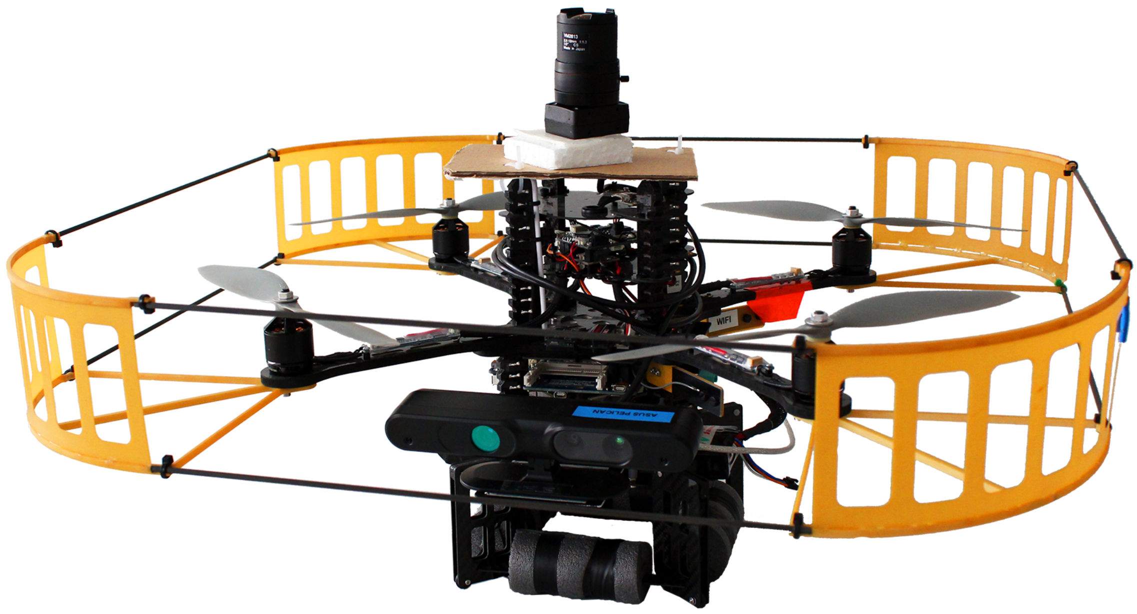


Figure 2.2.: AscTec Pelican quadrocopter with payload: the Autopilot Board, the Mastermind Board, a monocular grayscale camera and an ASUS Xtion PRO LIVE RGB-D camera.

The upper processor board is the *AutoPilot Board*. It has two ARM based processors, the so called HighLevel (HLP) and LowLevel processor (LLP). The LowLevel processor is a closed system and runs basic data fusion and control algorithms. The HighLevel processor is freely programmable. The lower processor board is the *Mastermind*. It is an embedded x86 PC based on an Intel Core2Duo processor with 1.86 GHz and 4 GB RAM. The operating system is a standard Ubuntu Linux distribution. A serial connection connects both boards for communication and data exchange.

The inertial measurement unit comprises a three axes gyroscope and accelerometer measuring the angular velocities and linear accelerations. The monocular grayscale camera is looking upwards and is used to track augmented reality markers at the ceiling to obtain an accurate, absolute position estimate for comparison and evaluation purposes.

For the control of the Pelican quadrocopter and on-board data fusion of multiple sensors the driver framework provided by Weiss et al. [55] is used. The data fusion is based on an extended Kalman filter (EKF), which estimates the full 6 degrees of freedom pose of the quadrocopter, several other parameters such as gyroscope and accelerometer bias, inter sensor pose, and scale.

The filtering framework consists of two main parts. One is the position control and EKF prediction loop running at 1 kHz on the HighLevel processor. The EKF prediction step integrates the IMU measurements. The second part of the framework runs on the Mastermind board. It implements the EKF correction step and provides the integration with the Robot Operating System (ROS) middleware. The EKF correction step corrects the integrated IMU measurements with more precise external pose or position measurements, however, only available at low rates (< 100 Hz). Such pose measurements can be obtained from visual odometry or a tracking system. The fused information is used to stabilize the quadrocopter in the air at a given position. The position of the Pelican can be changed by commanding velocities or absolute positions.

2.2. RGB-D Cameras

RGB-D cameras refer to digital cameras providing color (RGB) and depth (D) information for every pixel in the image. The depth information is obtained by different stereo vision techniques.

2.2.1. General

An RGB-D camera comprises a digital camera taking color images and additional devices measuring the depth of the scene. The color images are typically encoded in the RGB color space, hence the name. For depth measurement different technologies are available. These can be categorized as active or passive.

Stereo cameras are passive depth sensors. They consist of two RGB cameras with a known transformation between each other. Correspondences between individual image points are computed for each image pair. Using the different positions of a point in the two images and the transformation between the cameras the depth of the point can be computed [46].

Active technologies not only observe light reflected or emitted by the scene, but also emit light into the scene. Therefore, they also work in environments with little or no external light. Common active depth measuring devices providing a two-dimensional depth map per measurement are based on either time of flight or a projective stereo approach.

Time of flight systems emit pulses of light and measure the time between emission and return of the reflected light. Taking into account the speed of light, the distance of a point to the sensor can be calculated [46].

Projective stereo based sensors operate similar to stereo cameras. However, instead of a second camera, a projector is used. It projects a known pattern of points into the scene. The pattern is observed by the camera and correspondences between the observed points and the points of the pattern are computed. From the position of corresponding points in the camera image and pattern, and using the known transformation between projector and camera, the depth of every scene point can be estimated.

Active depth sensors often emit infrared (IR) light so it is not visible to humans. This can lead to missing measurements in outdoor environments, because the sun is a major source of infrared light blooming the projected pattern.

Lately, RGB-D cameras based on the projective stereo technology developed by PrimeSense, like the Microsoft Kinect and the ASUS Xtion PRO LIVE , have been used for visual odometry on a quadrocopter [24], 3D reconstruction [40] and handheld SLAM [12]. They are cheap, have low power consumption, and are light-weight compared to other stereo or time of flight cameras. Another advantage is, that they perform the depth computation on the camera.

These sensors have two cameras and a projector. Figure 2.3 depicts the ASUS Xtion PRO LIVE showing the setup of the components. One of the cameras gathers the RGB images.

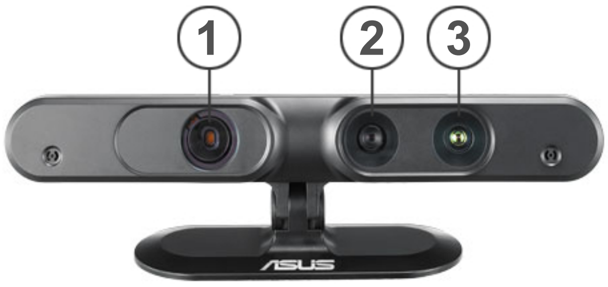


Figure 2.3.: ASUS Xtion PRO LIVE camera: ① IR projector ② RGB camera ③ IR camera [26]

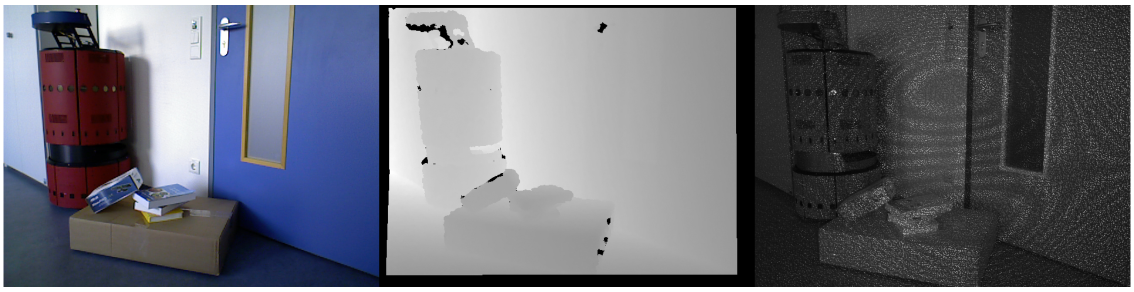


Figure 2.4.: RGB image, registered depth image and IR image of a scene obtained with an ASUS Xtion PRO LIVE .

The other one is an infrared camera observing the infrared light pattern emitted by the projector.

Figure 2.4 shows examples of an RGB image, a depth image and an IR image with the projected pattern. In the depth image brighter pixels indicate points further away, darker pixels points closer to the camera. Black pixels indicate missing depth values. These invalid pixels occur if no depth can be determined, e.g. the infrared pattern is not visible on reflective surfaces.

2.2.2. ASUS Xtion PRO LIVE

The ASUS Xtion PRO LIVE (see figure 2.3) is based on the projective stereo technology developed by PrimeSense. It provides RGB and depth images with VGA resolution (640×480 pixels) at a rate of 30 frames per second. A higher frame rate of 60 frames per second can be obtained by reducing the resolution to QVGA (320×240 pixels) [26].

The depth values are encoded as 16 bit unsigned integer values representing the depth in millimeters. According to the specification [26] the depth values range from 0.8 m to 3.5 m. In experiments measurements in the range of 0.7 m up to 9.5 m have been obtained. A detailed discussion of the data accuracy and calibration of PrimeSense RGB-D cameras, in particular the Kinect, is given in [29].

Compared to the Microsoft Kinect the ASUS Xtion PRO LIVE has the following advantages. The RGB and depth images are time synchronized and can be registered to each other on-board the camera. Furthermore, the camera has only a weight of ~150 g (Mi-

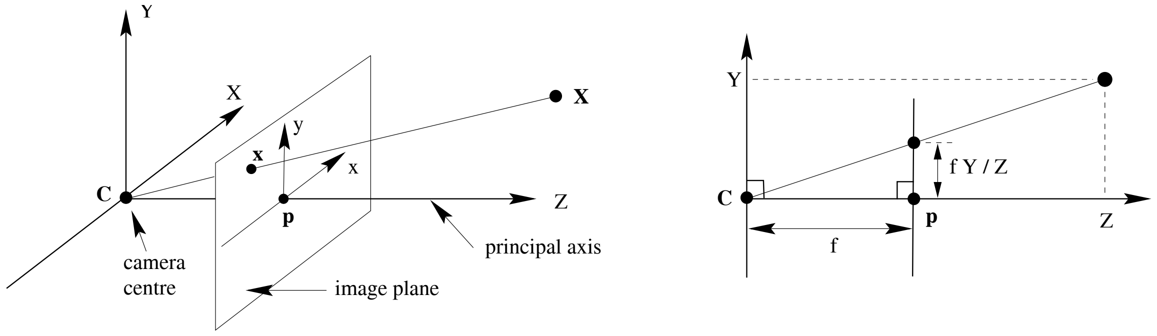


Figure 2.5.: Pinhole camera model [22].

crosoft Kinect ~440 g) and only needs the USB connection as power supply.

Depth registration causes invalid pixels at the border of the depth image (cf. figure 2.4), because the depth image has to be transformed into the viewpoint of the RGB camera to associate every color pixel with a depth pixel.

2.3. Camera Model

The camera model describes the mapping of points in the three-dimensional world to the two-dimensional image created by the camera [22]. This mapping π from 3D coordinates to 2D coordinates is named projection, and denoted by

$$\pi\colon \mathbb{R}^3 \to \mathbb{R}^2. \tag{2.1}$$

In this thesis the simple pinhole camera model, depicted in figure 2.5, is used. It abstracts the whole camera structure to an infinitely small hole, the pinhole, and an image plane. The projection described by the model is known as perspective projection. Only light rays falling through the hole and intersecting with the image plane get projected. The location of the pinhole is the optical center $\mathbf{C}$ of the camera. The distance between optical center and image plane is the focal length f.

In reality the image plane is located behind the optical center and not in front of it, in the model this can be neglected without loss of generality. Every 3D point, whose connection line to the optical center intersects the image plane gets represented in the image. The 3D coordinates (X, Y, Z) of the point $\boldsymbol{X}$ are related to its 2D image point $\boldsymbol{x}$ with coordinates (x, y) by the following equations:

$$\pi(X, Y, Z) \to (x, y) \tag{2.2}$$

$$x = \frac{X \cdot f_x}{Z} + o_x \tag{2.3}$$

$$y = \frac{Y \cdot f_y}{Z} + o_y \tag{2.4}$$

Since the pixels on the sensor chip do not have to be quadratic, two different focal lengths f_x and f_y are used. They are obtained through different scaling factors s_x and

s_y in the respective direction, thus $f_x = s_x \cdot f$ and $f_y = s_y \cdot f$. The offsets o_x and o_y account for the fact that the optical center $\mathbf{C}$ does not coincide with the origin of the image coordinates.

If the depth Z and the camera parameters are known the 3D point of an image point $\boldsymbol{x}$ can be reconstructed using the inverse of equation 2.2.

$$\pi^{-1}(x, y, Z) \mapsto (X, Y, Z) \tag{2.5}$$

$$X = \frac{x - o_x}{f_x} Z \tag{2.6}$$

$$Y = \frac{y - o_y}{f_y} Z \tag{2.7}$$

$$Z = Z \tag{2.8}$$

The intrinsic parameters (f_x, f_y, o_x, o_y) can be obtained by a standard camera calibration procedure [61]. During calibration further parameters like sensor skew or radial distortion coefficients are determined. The sensor skew accounts for a sensor not mounted parallel to the camera lens. The radial distortion coefficients model the distortion of the image due to the lens. Before the camera images are processed by the presented algorithms, these effects are removed. Therefore, the pinhole camera model holds.

2.4. Rigid Body Motion

This section covers the properties and representation of the motion of rigid bodies in three-dimensional space. The motion of a rigid body in three-dimensional Euclidean space preserves the distance and orientation between any pair of points on the object.

More formally a rigid body motion is a map g

$$g\colon \mathbb{R}^3 \to \mathbb{R}^3; \quad \mathbf{x} \mapsto g(\mathbf{x}) \tag{2.9}$$

preserving the distance and the orientation between two points $\boldsymbol{p}$ and $\boldsymbol{q}$

$$\|\boldsymbol{p} - \boldsymbol{q}\| = \|g(\boldsymbol{p}) - g(\boldsymbol{q})\| \qquad \forall\, \boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^3, \tag{2.10}$$

$$g(\boldsymbol{p}) \times g(\boldsymbol{q}) = g(\boldsymbol{p} \times \boldsymbol{q}) \qquad \forall\, \boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^3. \tag{2.11}$$

Such a map is called a special Euclidean transformation. The collection of all these transformations in three-dimensional Euclidean space forms the special Euclidean group $\mathrm{SE}(3)$.

The properties of distance and orientation preservation can be used to represent the motion of a rigid body in a compact way. The transformation of one point having an attached coordinate frame is sufficient to specify the motion of the whole object.

This transformation is always with respect to some reference coordinate frame and can be decomposed into a rotational and a translational part. The rotation changes the orientation of the object coordinate frame and the translation moves it in space. A rigid body motion has six degrees of freedom in total, three degrees for rotation and three degrees for translation.

For the rotational part of a transformation various representations exist [10]. A very common representation is a 3×3 orthogonal matrix R, the rotation matrix. All rotation matrices belong to the special orientation group $\mathrm{SO}(3)$. Other frequently used representations are quaternions, and a combination of a rotation angle and axis. The translation is represented as vector $\boldsymbol{t} \in \mathbb{R}^3$. The components of $\boldsymbol{t}$ specify the translation along the x, y and z axis. A rigid body motion g is a combination of a rotation matrix from SO(3) and a translation vector from $\mathbb{R}^3$. It can be expressed as a 4×4 matrix G

$$G = \begin{bmatrix} R & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \tag{2.12}$$

its inverse being

$$G^{-1} = \begin{bmatrix} R^T & -R^T\boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}. \tag{2.13}$$

The transformation of a point $\boldsymbol{p}$ with homogenous coordinates $(x, y, z, 1)^T$ using g can be expressed as matrix multiplication

$$g(\boldsymbol{p}) = g(G, \boldsymbol{p}) = G \cdot \boldsymbol{p} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.14}$$

i.e. with $\boldsymbol{p}' = g(\boldsymbol{p}) = (x', y', z', 1)^T$

$$\begin{aligned} x' &= r_{11}x + r_{12}y + r_{13}z + t_x, \\ y' &= r_{21}x + r_{22}y + r_{23}z + t_y, \\ z' &= r_{31}x + r_{32}y + r_{33}z + t_z. \end{aligned} \tag{2.15}$$

Multiple rigid body motions can be chained using the matrix representation and by left multiplying consecutive transformations. The identity transformation, meaning no motion, is given by $R = I$ and $\boldsymbol{t} = \boldsymbol{0}$, i.e. $g_{\text{identity}}(\boldsymbol{p}) = \boldsymbol{p}$.

The translation representation as vector $\boldsymbol{t}$ is canonical, because the three components of the vector are equal to the degrees of freedom. In contrast, the representation of a rotation as matrix is not canonical. It has nine parameters but only three degrees of freedom. The remaining parameters are constrained due to the orthogonality requirement of the matrix and the restriction to a norm equal to one for all row and column vectors. This imposes six constraints leaving only three free parameters.

A minimal representation for a rigid body motion g can be obtained by using the parameters of its associated Lie algebra $\mathfrak{se}(3)$. Such a minimal representation is useful when determining the parameters through a numerical optimization algorithm. Every transformation matrix in the Lie group SE(3) describing a rigid body motion has a representation in its associated Lie algebra with a 6×1 parameter vector $\boldsymbol{\xi} = (\boldsymbol{\nu}^T, \boldsymbol{\omega}^T)^T$ where $\boldsymbol{\nu} = (\nu_1, \nu_2, \nu_3)^T$ is the translational velocity and $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^T$ the rotational velocity.

The rigid body motion g can be calculated from its Lie algebra parameters $\boldsymbol{\xi}$ using the exponential map

$$\exp\colon \mathfrak{se}(3) \to \mathrm{SE}(3); \boldsymbol{\xi} \mapsto g, \tag{2.16}$$

$$G(\boldsymbol{\xi}) = e^{\hat{\boldsymbol{\xi}}} \tag{2.17}$$

where $\hat{\boldsymbol{\xi}}$ is known as twist and is the following 4×4 matrix

$$\hat{\boldsymbol{\xi}} = \begin{bmatrix} [\boldsymbol{\omega}]_\times & \boldsymbol{\nu} \\ \mathbf{0} & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & \nu_1 \\ \omega_3 & 0 & -\omega_1 & \nu_2 \\ -\omega_2 & \omega_1 & 0 & \nu_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{2.18}$$

The operator $[\boldsymbol{x}]_\times$ creates a 3×3 skew symmetric matrix from a 3×1 vector $\boldsymbol{x} = (x, y, z)^T$, i.e.

$$[\boldsymbol{x}]_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}. \tag{2.19}$$

The matrix exponential $e^{\hat{\boldsymbol{\xi}}}$ has a closed form solution [38, 49]

$$e^{\hat{\boldsymbol{\xi}}} = \begin{bmatrix} e^{[\boldsymbol{\omega}]_\times} & V\boldsymbol{\nu} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} R & \boldsymbol{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.20}$$

where $e^{[\boldsymbol{\omega}]_\times}$ is computed using Rodrigues' formula

$$e^{[\boldsymbol{\omega}]_\times} = I + \frac{\sin(\|\boldsymbol{\omega}\|)}{\|\boldsymbol{\omega}\|}[\boldsymbol{\omega}]_\times + \frac{1 - \cos(\|\boldsymbol{\omega}\|)}{\|\boldsymbol{\omega}\|^2}[\boldsymbol{\omega}]_\times^2 \tag{2.21}$$

and V is

$$V = I + \frac{1 - \cos(\|\boldsymbol{\omega}\|)}{\|\boldsymbol{\omega}\|^2}[\boldsymbol{\omega}]_\times + \frac{\|\boldsymbol{\omega}\| - \sin(\|\boldsymbol{\omega}\|)}{\|\boldsymbol{\omega}\|^3}[\boldsymbol{\omega}]_\times^2. \tag{2.22}$$

The inverse to the exponential map is called the logarithm map

$$\log\colon \mathrm{SE}(3) \to \mathfrak{se}(3); g \mapsto \boldsymbol{\xi}, \tag{2.23}$$

$$\boldsymbol{\xi} = \log(G). \tag{2.24}$$

The identity transformation is obtained for $\boldsymbol{\xi} = \mathbf{0}$.

2.5. Least Squares

Least squares is a common technique to estimate parameters of a model from noisy observations. The following sections describe the general idea and formulas, how these can be derived from a probabilistic point of view and the extension to the weighted least squares method.

2.5.1. General

Least squares fits a model $f(\boldsymbol{x}, \boldsymbol{\theta})$ with a fixed set of parameters $\boldsymbol{\theta}$ $(\theta_0, \theta_1, \ldots, \theta_{m-1})^T$ for all $\boldsymbol{x}$ $(x_0, x_1, \ldots, x_{n-1})^T$ to observations $\hat{\boldsymbol{f}}$ $(\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_{n-1})^T$ by minimizing the squared error between the observations and the model. The error is not guaranteed to be zero, because the observations typically do not exactly fit the model due to noise or a simplified model. This can be formalized as

$$E_{\text{LS}}(\theta) = (\hat{\boldsymbol{f}} - f(\boldsymbol{x}, \boldsymbol{\theta}))^T(\hat{\boldsymbol{f}} - f(\boldsymbol{x}, \boldsymbol{\theta})) = \sum_i^n (\hat{f}_i - f(x_i, \boldsymbol{\theta}))^2 = \sum_i^n (r_i(\boldsymbol{\theta}))^2 \tag{2.25}$$

where $r_i(\boldsymbol{\theta}) = \hat{f}_i - f(x_i, \boldsymbol{\theta})$ is the i^{th} residual. The best parameters $\boldsymbol{\theta}_{\text{LS}}$ are obtained by solving

$$\boldsymbol{\theta}_{\text{LS}} = \arg\min_{\boldsymbol{\theta}} E_{\text{LS}}(\boldsymbol{\theta}). \tag{2.26}$$

The minimum is found by calculating the partial derivatives of E_{LS} and setting them equal to zero.

$$\frac{\partial E_{\text{LS}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_i^n \frac{\partial r_i(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} 2 r_i(\boldsymbol{\theta}) = 0 \tag{2.27}$$

There have to be at least as many observations as unknowns in $\boldsymbol{\theta}$ to find a solution. In practice, there are much more observations than unknowns leading to an overconstrained equation system.

In the special case where $f(x_i, \boldsymbol{\theta})$ is linear in the parameters $\boldsymbol{\theta}$, i.e. $f_{\text{lin}}(x_i, \boldsymbol{\theta}) = A(x_i)^T \boldsymbol{\theta}$, solving equation 2.26 is called linear least squares. $A(x_i)$ is an arbitrary function of x_i of dimension m. The solution for linear least squares can be obtained in closed form. Plugging $f_{\text{lin}}(x_i, \boldsymbol{\theta})$ into equation 2.27 leads to

$$\sum_i^n A(x_i)^T \left(\hat{f}_i - A(x_i)\boldsymbol{\theta}\right) = 0. \tag{2.28}$$

This equation can be rearranged to

$$\sum_i^n A(x_i)^T A(x_i) \boldsymbol{\theta} = \sum_i^n A(x_i)^T \hat{f}_i \tag{2.29}$$

and rewritten in matrix notation

$$A(\boldsymbol{x})^T A(\boldsymbol{x}) \boldsymbol{\theta} = A(\boldsymbol{x}) \hat{\boldsymbol{f}}. \tag{2.30}$$

The parameter vector $\boldsymbol{\theta}$ is the solution to these normal equations.

For models $f(x_i, \boldsymbol{\theta})$ with non-linear dependence on $\boldsymbol{\theta}$ the technique is called non-linear least squares. To obtain a linear dependence on the parameters $\boldsymbol{\theta}$, $r_i(\boldsymbol{\theta})$ has to be linearized using a first order Taylor expansion at a point $\boldsymbol{\alpha}$ where $\boldsymbol{\theta} = \boldsymbol{\alpha}$. The linearized residual is

$$r_{\text{lin},i}(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\alpha}} = r_i(\boldsymbol{\alpha}) + \left.\frac{\partial r_i(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\alpha}} (\boldsymbol{\theta} - \boldsymbol{\alpha}) = r_i(\boldsymbol{\alpha}) + J_i(\boldsymbol{\alpha})(\boldsymbol{\theta} - \boldsymbol{\alpha}), \tag{2.31}$$

its derivative is

$$\left.\frac{\partial r_{\text{lin},i}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\alpha}} = J_i(\boldsymbol{\alpha}). \tag{2.32}$$

Because equation 2.31 is only an approximation of $r_i(\boldsymbol{\theta})$ at $\boldsymbol{\theta} = \boldsymbol{\alpha}$ the solution has to be obtained iteratively. In each iteration k an increment $\Delta\boldsymbol{\theta}$ to the solution is computed. The solution at a certain iteration $k+1$ is given as $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}$. At every iteration the residual is re-linearized around the last solution

$$r_{\text{lin},i}(\boldsymbol{\theta}_{k+1})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k} = r_i(\boldsymbol{\theta}_k) + J_i(\boldsymbol{\theta}_k)(\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k) = r_i(\boldsymbol{\theta}_k) + J_i(\boldsymbol{\theta}_k)\Delta\boldsymbol{\theta}. \tag{2.33}$$

The increments are calculated by plugging equation 2.33 and equation 2.32 with $\boldsymbol{\alpha} = \boldsymbol{\theta}_k$ into equation 2.27 and solving the resulting equation system

$$\sum_i^n J_i(\boldsymbol{\theta}_k)^T 2\Big(r_i(\boldsymbol{\theta}_k) + J_i(\boldsymbol{\theta}_k)\Delta\boldsymbol{\theta}\Big) = 0, \tag{2.34}$$

$$\sum_i^n J_i(\boldsymbol{\theta}_k)^T J_i(\boldsymbol{\theta}_k)\Delta\boldsymbol{\theta} = -\sum_i^n J_i(\boldsymbol{\theta}_k)^T r_i(\boldsymbol{\theta}_k). \tag{2.35}$$

Putting this into matrix notation the normal equations (cf. equation 2.30) of non-linear least squares are obtained

$$J(\boldsymbol{\theta}_k)^T J(\boldsymbol{\theta}_k)\Delta\boldsymbol{\theta} = -J(\boldsymbol{\theta}_k)^T \boldsymbol{r}(\boldsymbol{\theta}_k) \tag{2.36}$$

where $J(\boldsymbol{\theta}_k)$ is the $n \times m$ Jacobian matrix with the $J_i(\boldsymbol{\theta}_k)$ as row vectors. The optimization technique of iteratively solving the normal equations for increments and re-linearizing the error at the new estimate is known as Gauss-Newton method. It takes advantage of the squared error term. Therefore, only first order approximations to the residuals $r_i(\boldsymbol{\theta})$ are required. The product of the transposed Jacobian with itself ($J^T J$) is, in fact, an approximation to the matrix of second order derivatives (Hessian matrix).

The Gauss-Newton algorithm only converges to the closest local minimum or might even diverge. The initial parameter estimate $\boldsymbol{\theta}_0$ should therefore be close to the true solution. There are extensions to the algorithm, like the Levenberg-Marquardt algorithm, improving the performance. Throughout this thesis only Gauss-Newton optimization is used.

2.5.2. Bayesian Derivation

From a Bayesian perspective the least squares approach can be derived by maximizing the a posteriori probability of the parameters $\boldsymbol{\theta}$ given the observations $\hat{\boldsymbol{f}}$

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \hat{\boldsymbol{f}}). \tag{2.37}$$

After applying Bayes' rule this becomes

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} \frac{p(\hat{\boldsymbol{f}} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\hat{\boldsymbol{f}})}. \tag{2.38}$$

$p(\hat{\boldsymbol{f}})$ can be neglected, because it does not depend on $\boldsymbol{\theta}$. Assuming a uniform distribution for $p(\boldsymbol{\theta})$, i.e. meaning all possible parameter vectors are equally likely, equation 2.38 simplifies to

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\hat{\boldsymbol{f}} \mid \boldsymbol{\theta}). \tag{2.39}$$

Further assuming that all observations are independent and identically distributed (iid.) equation 2.39 can be written as

$$\boldsymbol{\theta}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\arg\max} \prod_{i}^{n} p(\hat{f}_i \mid \boldsymbol{\theta}). \tag{2.40}$$

Instead of maximizing the likelihood the negative log-likelihood is minimized, because it simplifies calculation if an exponential term is present,

$$\boldsymbol{\theta}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\arg\min} \sum_{i}^{n} -\log(p(\hat{f}_i \mid \boldsymbol{\theta})). \tag{2.41}$$

In case $p(\hat{f}_i \mid \boldsymbol{\theta})$ is normal distributed $\mathcal{N}(\hat{f}_i, \mu, \sigma)$ with mean $\mu = f(x_i, \boldsymbol{\theta})$ and standard deviation σ equation 2.41 reduces to

$$\boldsymbol{\theta}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\arg\min} \sum_{i}^{n} \frac{1}{2\sigma^2} \left(\hat{f}_i - f(x_i, \boldsymbol{\theta})\right)^2 = \underset{\boldsymbol{\theta}}{\arg\min} \sum_{i}^{n} \frac{1}{2\sigma^2} \left(r_i(\boldsymbol{\theta})\right)^2, \tag{2.42}$$

being equivalent to equation 2.26, because the constant scale factor $\frac{1}{2\sigma^2}$ can be dropped. The same formulation can be obtained by using $p(r_i \mid \boldsymbol{\theta}) = \mathcal{N}(r_i(\boldsymbol{\theta}), 0, \sigma)$ instead of $p(\hat{f}_i \mid \boldsymbol{\theta}) = \mathcal{N}(\hat{f}_i, f(x_i, \boldsymbol{\theta}), \sigma)$.

In equation 2.38 prior knowledge about the parameters $\boldsymbol{\theta}$ can be incorporated with the distribution $p(\boldsymbol{\theta})$. If $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}, \boldsymbol{\theta}_{\text{prior}}, \sigma_{\text{prior}})$ equation 2.42 turns into

$$\boldsymbol{\theta}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\arg\min} \left(\sum_{i}^{n} \frac{1}{2\sigma^2} \left(r_i(\boldsymbol{\theta})\right)^2 \right) + \frac{1}{2\sigma_{\text{prior}}^2} \left(\boldsymbol{\theta} - \boldsymbol{\theta}_{\text{prior}}\right)^2. \tag{2.43}$$

The extended normal equations (cf. equation 2.36) with prior on the parameters are

$$\left(J(\boldsymbol{\theta}_k)^T J(\boldsymbol{\theta}_k) + \lambda I\right) \Delta\boldsymbol{\theta} = -J(\boldsymbol{\theta}_k)^T \boldsymbol{r}(\boldsymbol{\theta}_k) + \lambda(\boldsymbol{\theta}_{\text{prior}} - \boldsymbol{\theta}_k) \tag{2.44}$$

where $\lambda = 1/\sigma_{\text{prior}}^2$ is a hyper-parameter controlling how close the solution will be to the prior. If the least squares problem is solved iteratively, estimates from previous iterations have to be subtracted from the prior.

2.5.3. Weighted Least Squares

The standard least squares formulation introduced so far has one drawback: Large values in $\boldsymbol{r}(\boldsymbol{\theta})$ heavily influence the parameter estimates, because of the quadratic term in equation 2.25. Often, these residuals are outliers caused by observations which cannot be explained by the model. To address this problem the quadratic term x^2 is replaced with a robust error function $\rho(x)$

$$\boldsymbol{\theta}_{\text{WLS}} = \underset{\boldsymbol{\theta}}{\arg\min}\ E_{\text{WLS}}(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\arg\min} \sum_{i}^{n} \rho(r_i(\boldsymbol{\theta})). \tag{2.45}$$

Comparing equation 2.45 to equation 2.41 yields that $\rho(x)$ should be the negative logarithm of the error probability distribution. In the robust statistics literature various robust error functions $\rho(x)$ have been proposed [25].

The minimum of equation 2.45 is found by deriving it with respect to $\boldsymbol{\theta}$ and setting the derivative equal to zero

$$\frac{\partial E_{\text{WLS}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{i}^{n} \frac{\partial r_i(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \psi(r_i(\boldsymbol{\theta})) = 0 \tag{2.46}$$

where $\psi(x)$ is the derivative of the robust error function known as influence function. Using the following alternative formulation with $\omega(x)x = \frac{\psi(x)}{x}x$

$$\frac{\partial E_{\text{WLS}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{i}^{n} \frac{\partial r_i(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \omega(r_i(\boldsymbol{\theta})) r_i(\boldsymbol{\theta}) = 0 \tag{2.47}$$

it becomes apparent that equation 2.46 is also the minimizer of the iteratively re-weighted least squares (IRLS) formulation [52]

$$E_{\text{IRLS}}(\boldsymbol{\theta}) = \sum_{i}^{n} \omega(r_i(\boldsymbol{\theta})) \Big(r_i(\boldsymbol{\theta}) \Big)^2 . \tag{2.48}$$

Comparing equation 2.47 to equation 2.27 it is obvious that each weight $\omega_i = \omega(r_i(\boldsymbol{\theta}))$ is just a scaling factor for the corresponding residual $r_i(\boldsymbol{\theta})$. Therefore, the weights can be incorporated into the normal equations (cf. equation 2.36) using a diagonal $n \times n$ matrix W with the diagonal elements $w_{ii} = \omega_i$

$$J(\boldsymbol{\theta}_k)^T W J(\boldsymbol{\theta}_k) \Delta\boldsymbol{\theta} = -J(\boldsymbol{\theta}_k)^T W \boldsymbol{r}(\boldsymbol{\theta}_k). \tag{2.49}$$

The computation of weights and new parameter estimates is repeated until convergence.

In case $p(r_i \mid \boldsymbol{\theta}) = \mathcal{N}(r_i(\boldsymbol{\theta}), 0, \sigma)$ the robust error function $\rho(x) = \frac{1}{2}(\frac{x}{\sigma})^2$, its influence function $\psi(x) = \frac{x}{\sigma^2}$, and the weight function $\omega(x) = \frac{1}{\sigma^2}$. When the distribution $p(r_i \mid \boldsymbol{\theta})$ has a scale parameter σ, which cannot be factored out of the weight function (in contrast to the normal distributed case), it has to be estimated as well. One problem is that the outliers also influence the scale estimate. Therefore, robust methods have to be used for the estimation of σ [25].

A common robust estimator for the scale is the Median Absolute Deviation (MAD). It is defined as:

$$\sigma_{\text{MAD}}(\boldsymbol{x}) = c \cdot \text{median}(|\boldsymbol{x} - \text{median}(\boldsymbol{x})|) \tag{2.50}$$

where $c = 1.4826$ is a scaling factor. In case the elements in $\boldsymbol{x}$ are drawn from a normal distribution, but also contain outliers, this scaling factor makes σ_{MAD} an approximation of the standard deviation of the distribution.

3. Related Work

Visual odometry refers to odometry using measurements from optical devices. Odometry is the estimation of the motion of a moving sensor from its data. It can be used for robot control and localization, when no external reference systems are available. In visual odometry the camera motion is estimated from the images gathered by the camera.

Camera motion estimation is a widely studied field and has various application areas. The variety of approaches can be split in two major categories *sparse* and *dense* visual odometry. Sparse visual odometry refers to approaches only using a part of the available image data, e.g. small patches at certain points in the image. Dense visual odometry in contrast uses the whole image data. Dense approaches became feasible in recent years with the advent of faster computational resources. Especially general-purpose computing on graphics processing units (GPGPU) allowing parallel processing of large datasets enabled many algorithms to run at high frame rates.

The following sections give an overview of state of the art sparse and dense visual odometry approaches.

3.1. Sparse Visual Odometry

Sparse visual odometry approaches have successfully been used to control a variety of robots, such as ground vehicles [42, 32] and more recently quadrocopters [24, 11, 56]. Engel et al. [11] and Weiss et al. [56] use the Parallel Tracking and Mapping (PTAM) system [31]. Huang et al. [24] use a similar system, but assess different options for each component of the processing pipeline and choose the best based on the trade-off between accuracy and runtime. All have in common that the visual odometry estimates are fused with measurements from the IMU mounted on the quadrocopter.

The typical pipeline in a sparse visual odometry system is the following: First, feature points are extracted from the new image using detectors like FAST [43] or Harris [21]. Afterwards, correspondences between the new features and features from the previous frame are established. This can be done by comparing small patches around the feature points. A match is assumed if the error between two patches is minimal. Instead of patches feature descriptors like SIFT [35] or SURF [4] can be used. These descriptors are computed from the surrounding pixels of a feature point and represented as vectors. While being more robust to mismatches than image patches, these descriptors are more expensive to compute. Finally, the transformation between two images is computed by minimizing the reprojection error between every pair of matched feature points. Various sophisticated techniques can be implemented to ensure correct feature association and improving the accuracy of the motion estimate. Besides a detailed overview of the visual odometry research in the past decades Fraundorfer and Scaramuzza [45, 13] provide a discussion of the various feature extractors and descriptors, and matching techniques for sparse visual odometry.

To estimate the rigid body motion of the camera the depth of the points has to be known, otherwise it is only possible to estimate a homography aligning the images. In case only a monocular camera is used, like in PTAM, at the beginning a stereo initialization has to be performed giving the depth for the first feature points. The depth of new points can subsequently be calculated by triangulation, when the camera motion is estimated from points with already known depth. Still the depth of the points cannot be determined in metric values, but just up to a scale factor. Engel et al. [11] and Weiss et al. [56] use the measurements from the IMU and an extended Kalman filter to simultaneously estimate the unknown scale. Using RGB-D cameras (as in [24]) the absolute depth is known, and the problem is simplified.

As complementary steps the detected features and camera poses can be integrated into a global map. Using global optimization techniques the map and the camera trajectory can be further tuned to obtain position estimates with higher precision and compensate the drift over time. These approaches are named simultaneous localization and mapping (SLAM).

3.2. Dense Visual Odometry

Dense visual odometry approaches, in contrast to sparse approaches, use the whole image data. They estimate the camera motion by aligning consecutive images through minimization of an error metric.

One of the first dense visual odometry methods was proposed by Comport et al. based on stereo image pairs [8]. Steinbrücker et al. [47] and Tykkälä et al. [54] recently proposed similar dense methods using the data from RGB-D cameras. All three approaches minimize the photometrical error between two images. This can be seen as extension of the 2D Lucas-Kanade image alignment algorithm [37] to 3D. In their article series Baker and Matthews [3] discuss various optimizations and extensions to the Lucas-Kanade algorithm in detail.

Alternatively, the geometrical error between 3D surfaces can be minimized instead of the photometrical error between images. These are so called iterative closest points (ICP) algorithms. Various variants exist [44]. One drawback is that they require structured 3D surfaces. Another disadvantage is that they involve a computationally expensive nearest neighbor search to create point correspondences. For small displacements this can be overcome using the projective lookup algorithm [7]. If the 3D surfaces are represented as 2D depth maps, the correspondence for one point in a second depth map is found by applying the rigid body motion and projecting it to 2D coordinates. These coordinates allow to simply look up the correspondence in the depth map by computing the memory address.

For speedup Henry et al. [23] extract features from the color images and perform ICP on the feature points and their corresponding 3D points computed from the depth map. Stückler et al. [50] transform every RGB-D image into a surfel octree where every node contains a Gaussian distribution modeling the color and point distribution. Feature descriptors are computed for all octree nodes. The alignment is done by establishing point correspondences between the features in two octrees and afterwards applying ICP. The octree representation naturally supports a coarse to fine alignment scheme over multiple

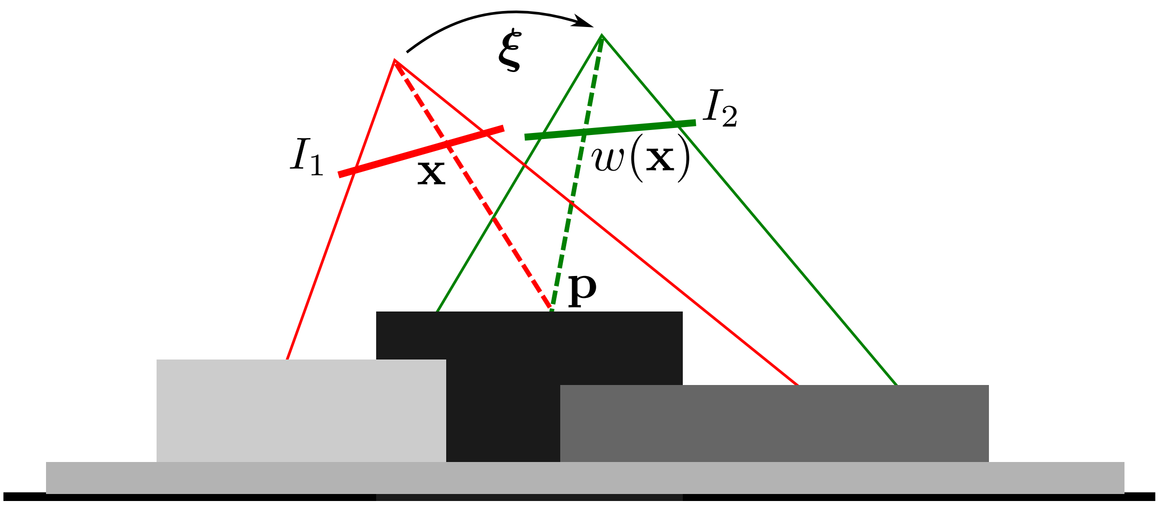


Figure 3.1.: The photo-consistency assumption: a point $\boldsymbol{p}$ is present in two images I_1 and I_2. The point is projected to the pixel $\boldsymbol{x}$ in I_1. Given the rigid body motion of the camera $\boldsymbol{\xi}$ the warping function $w(\boldsymbol{x})$ computes the pixel coordinates in I_2. The intensity values should be the same given the correct $\boldsymbol{\xi}$, i.e. $I_1(\boldsymbol{x}) - I_2(w(\boldsymbol{x})) = 0$.

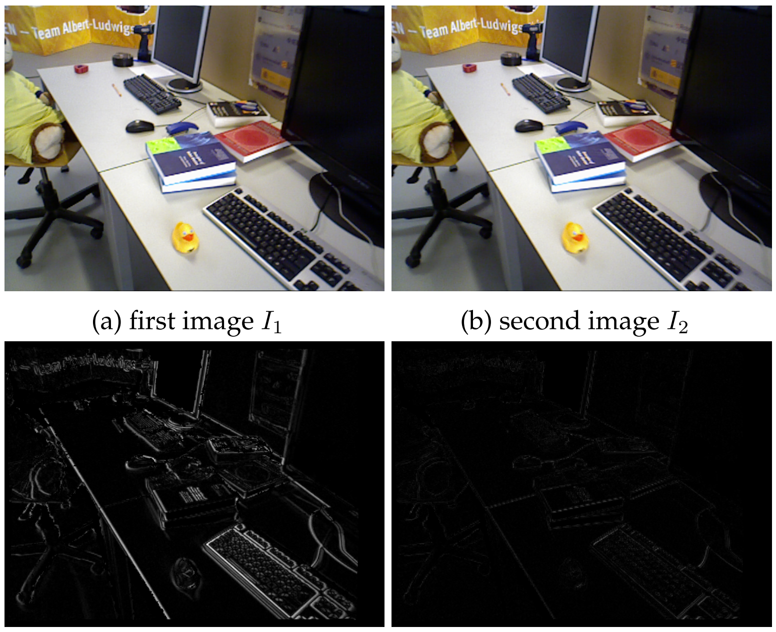
resolutions. These approaches are similar to the described sparse approaches (cf. 3.1).

In general, all odometry methods that accumulate the motion estimates between consecutive frames suffer from long-term drift [27]. Therefore, KinectFusion [40] and Dense Tracking and Mapping (DTAM) [41] build, parallel to tracking the camera, a global model of the environment. The camera is tracked by registering the current image to synthesized views generated from the model. Each new frame is incorporated into the model. In DTAM only a monocular camera is used and the model is required to estimate the 3D structure of the scene [41]. While KinectFusion [40] employs a variant of the ICP algorithm to align surfaces obtained from a RGB-D camera to the model, DTAM [41] utilise a photometrical error similar to Steinbrücker et al. Realtime performance of these model based approaches is achieved through general-purpose computing on GPUs.

So far, none of the dense approaches has been applied on a quadrocopter for stabilization and position control. In this thesis an approach build upon the recent work of Steinbrücker et al. and Tykkälä et al. utilizing the data from an RGB-D sensor is employed for the control of a quadrocopter, because it is:

- highly accurate compared to sparse, feature-based and ICP-based methods,
- much simpler than feature-based approaches involving many intermediary steps and fine tuning,
- fast to compute on a CPU and therefore suited for the limited computational resources available on a quadrocopter in contrast to model based approaches.

The general idea behind these approaches is that a point on a 3D surface observed by multiple cameras has the same intensity in all images, which is the so called photo-consistency assumption. In case the rigid body motion between the cameras is known,


(a) first image I_1 (b) second image I_2

(c) residuals before warping I_2 (d) residuals after warping I_2

Figure 3.2.: Two consecutive RGB images (a, b) and the photometrical error between them before warping the second image I_2 towards I_1 (c) and after warping it with the correct transformation (d).

the image of a point in one camera can be transformed into the image from the point of view of another camera. The error between the two images of point $\boldsymbol{p}$ should be zero. This idea is illustrated in figure 3.1. In practice the error over all image points will not be zero, because of occlusions, dynamic objects, illumination changes, and sensor noise. Nevertheless, the correct rigid body motion should lead to the lowest overall error. Figure 3.2 shows two consecutive images and two error images. The first error image (see figure 3.2c) is calculated without warping the second image. The second error image (see figure 3.2d) has been computed after warping the second image with the correct transformation. The transformation has been obtained from a precise external tracking system.

Using this principle the unknown rigid body motion between two known camera images can be found as the one minimizing the error between the images. As preliminary experiments showed the approach of Steinbrücker et al. has some limitations:

- wrong estimates when dynamic objects, like persons, move through the scene,
- occasional jumps in the estimated trajectory,
- execution speed still not sufficient for application on quadrocopters.

In this thesis, these limitations are overcome by employing methods from robust and Bayesian statistics [25, 15]. Further, an effective minimization scheme and the use of SIMD instructions accelerate the implementation.

3.3. RGB-D Benchmark

The RGB-D benchmark developed by Sturm et al. [51] provides a method to evaluate algorithms utilizing RGB-D images, e.g. visual odometry, SLAM, or 3D reconstruction approaches. The benchmark consists of several RGB-D image sequences. One set of sequences contains ground truth data of the camera motion obtained by a precise external tracking system. A second set intended for validation has no publicly available ground truth data. The sequences are grouped into different categories corresponding to different application scenarios, e.g. SLAM from a handheld camera or a wheeled robot, object reconstruction, and robustness against dynamic objects.

Additionally, the RGB-D benchmark contains different tools to compute quality metrics for an estimated trajectory in comparison to the ground truth. These metrics are the absolute trajectory error and the relative pose error. The absolute trajectory error measures the distance between the true and the estimated endpoint. This is useful to evaluate the performance of SLAM algorithms. The relative pose error computes the translational or rotational drift of the estimate with respect to the ground truth over a certain temporal distance, i.e. the drift per frame or the drift per second. This metric is suitable for the evaluation of the performance of visual odometry approaches. For each metric the Root Mean Square Error (RMSE), mean, median, standard deviation, minimum, and maximum value can be computed. A drawback of these diverse options is that no standard metric has been established yet, i.e. different authors measure the median drift per frame and others the RMSE drift per second.

Using the RGB-D benchmark for evaluation has the advantage that it facilitates the objective comparison of different approaches. The large number of datasets with varying scene content ensures a good generalization of the tested approach. It also frees authors from the tedious task of recording real world datasets with ground truth.

4. Approach

4.1. Overview

The approach is based on the photo-consistency assumption between images captured by the camera. Given the correct motion of the camera between two images they can be transformed into the same viewpoint and the intensity difference over all pixels is zero under the Lambertian assumption. In reality the difference, or error, will never be zero, because of occlusions, dynamic objects, sensor noise and other effects. Still the error is supposed to be minimal between the images given the correct camera motion.

Applying this principle the unknown camera motion (with parameters $\boldsymbol{\xi}^*$) can be found by minimizing the intensity error between two images. This can be formalized as

$$\boldsymbol{\xi}^* = \arg\min_{\boldsymbol{\xi}} \, (I_2(w(\boldsymbol{\xi}, \boldsymbol{x})) - I_1(\boldsymbol{x}))^2 \tag{4.1}$$

with $w(\boldsymbol{\xi}, \boldsymbol{x})$ being the *warping* function defined as

$$w(\boldsymbol{\xi}, \boldsymbol{x}) = \pi(g(G(\boldsymbol{\xi}), \pi^{-1}(\boldsymbol{x}, Z_1(\boldsymbol{x})))). \tag{4.2}$$

It transforms the reconstructed 3D point of $\boldsymbol{x}$ obtained by $\pi^{-1}(\boldsymbol{x}, Z_1(\boldsymbol{x}))$ (cf. equation 2.5) with the rigid body motion $G(\boldsymbol{\xi})$. Afterwards, it projects the transformed point (see equation 2.2) onto the image plane of the second camera, again.

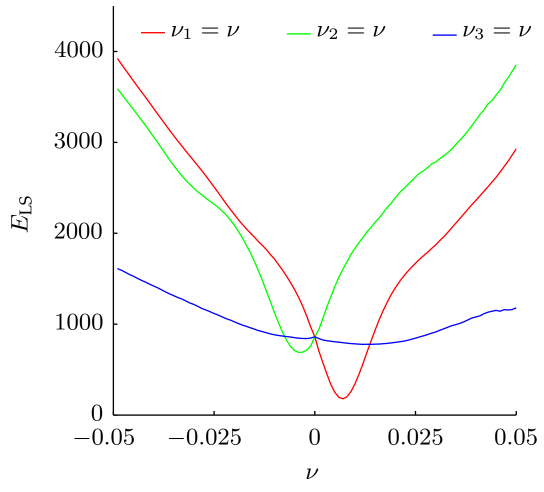
Equation 4.1 can be solved using the least squares approach (see section 2.5), because it has the form of equation 2.26 with $\boldsymbol{\theta} = \boldsymbol{\xi}$ and $E_{\text{LS}}(\boldsymbol{\xi}) = (I_2(w(\boldsymbol{\xi}, \boldsymbol{x})) - I_1(\boldsymbol{x}))^2$, i.e.

$$\boldsymbol{\xi}^* = \boldsymbol{\xi}_{\text{LS}} = \arg\min_{\boldsymbol{\xi}} E_{\text{LS}}(\boldsymbol{\xi}) = \arg\min_{\boldsymbol{\xi}} (I_2(w(\boldsymbol{\xi}, \boldsymbol{x})) - I_1(\boldsymbol{x}))^2. \tag{4.3}$$

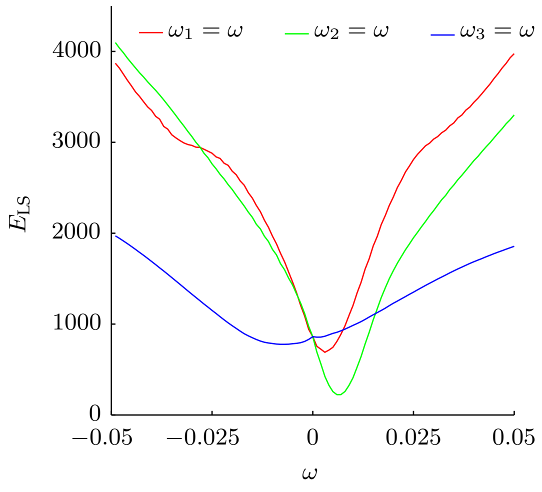
This is a non-linear least squares problem, because $w(\boldsymbol{\xi}, \boldsymbol{x})$ is non-linear. Nevertheless, the error function often has a distinct global minimum as shown by the plots of the error in figure 4.1. The error plots are created by computing the error between I_1 (see figure 3.2a) and the warped I_2 (see figure 3.2b). For figure 4.1a I_2 is warped separately along the x (red), y (green) and z (blue) axis by varying the corresponding value ν while fixing the other values in $\boldsymbol{\xi}$ to zero. The translation is in the range of ± 5 cm. Similarly, I_2 is warped by separately rotating around the x (red), y (green) and z (blue) axis by changing the respective ω value for figure 4.1b. The rotation is between ± 0.05 rad ($\pm 2.9°$).

The error graphs for translation along x and y axis, and respectively rotation around these axis, are very steep. In contrast the error for transformation with respect to the z axis is flat.

Figure 4.2 shows the error for translation along the different axis computed at different resolutions of the two images, ranging from 640×480 to 80×60 pixels. At all resolutions there is only one distinct minimum per error curve and it is at the same position. A notable



(a) E_{LS} after translation of I_2



(b) E_{LS} after rotation of I_2

Figure 4.1.: E_{LS} after I_2 is warped by varying the components of $\boldsymbol{\xi}$ corresponding to translation and rotation around x, y and z axis. Although, the warping function is non-linear the graphs have distinct minima. The graphs are only convex close to the minima.

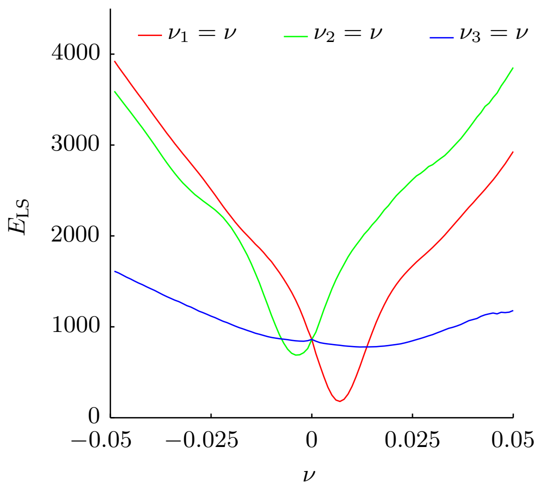
difference between the resolutions is, that the area around the minimum flattens out at lower resolutions. While the optimization is faster at lower resolutions the solution is more precise at higher resolutions.

4.2. Linearization

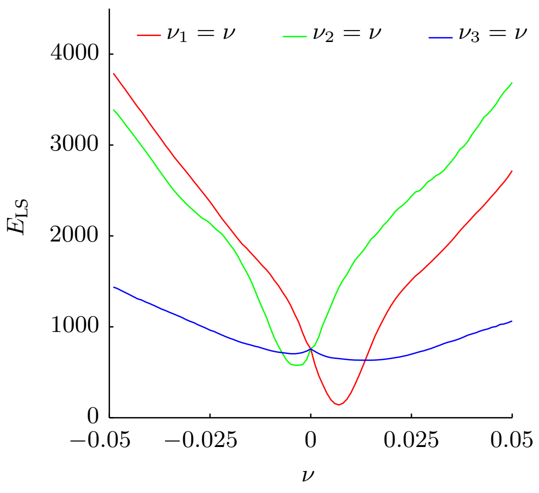
The error term in equation 4.3 is not linear which gives rise to a non-linear least squares problem. As described in section 2.5 such problems can be solved by linearizing them around a point using Taylor expansion and solving the resulting linear least squares problem. This is repeated using the latest estimate until it converges to a solution. Figure 4.3 visualizes the different components of the Jacobian computed for linearization. The values of the Jacobian indicate the change of the error when translating along or rotating around the respective component of $\boldsymbol{\xi}$. Baker and Matthews showed that multiple equivalent formulations for the linearization of image alignment problems exist [3]. In the following, two linearization variants are described to solve equation 4.3.

The first variant re-linearizes around the last estimate at every iteration and is the application of the general equations from section 2.5 to equation 4.3. Baker and Matthews name this the *Forward Additive Algorithm* [3].

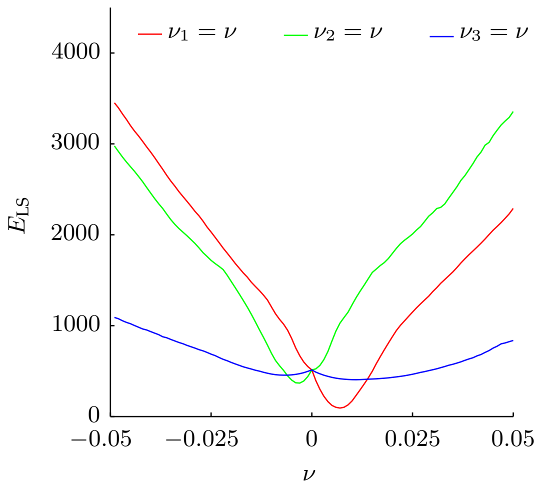
The second variant only requires linearization around one fixed point, typically the identity. This is achieved by *restarting* the algorithm after every iteration just updating the input images with the last estimate. Therefore, only linearization around the initial point is required. This saves computational cost and simplifies the analytical Jacobians. Baker and Matthews call this the *Forward Compositional Algorithm* [3].



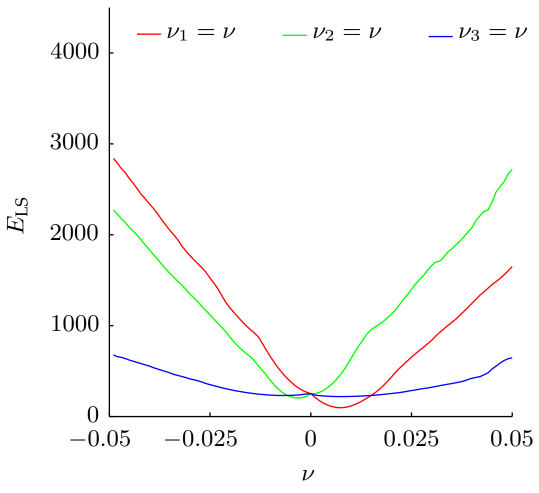
(a) E_{LS} for 640×480 pixels



(b) E_{LS} for 320×240 pixels

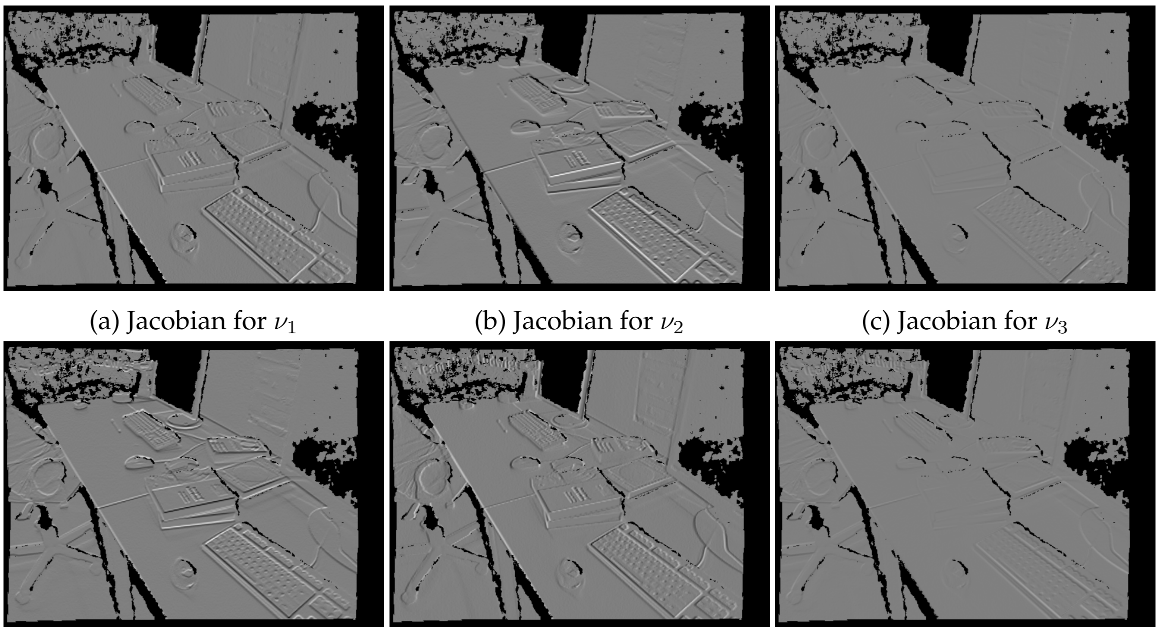


(c) E_{LS} for 160×120 pixels



(d) E_{LS} for 80×60 pixels

Figure 4.2.: The error E_{LS} computed for translations of I_2 at different resolutions. The overall shape and the minima of the graphs are the same on all resolutions. The convex area grows for lower resolutions.



(a) Jacobian for ν_1 (b) Jacobian for ν_2 (c) Jacobian for ν_3

(d) Jacobian for ω_1 (e) Jacobian for ω_2 (f) Jacobian for ω_3

Figure 4.3.: Jacobians for the different components of $\boldsymbol{\xi}$. The values indicate the change of the error when modifying the respective component. The medium gray (■) represents 0, brighter pixels positive values and darker pixels negative values. Black pixels are invalid.

4.2.1. Forward Additive Algorithm

The Forward Additive Algorithm solves the non-linear image alignment problem by iteratively computing estimates of the transformation parameters $\boldsymbol{\xi}$. The parameter increments $\Delta\boldsymbol{\xi}$ are concatenated with the previous estimate using $\boldsymbol{\xi}_{k+1} = \log(\exp(\boldsymbol{\xi}_k) \cdot \exp(\Delta\boldsymbol{\xi})) = \boldsymbol{\xi}_k \boxplus \Delta\boldsymbol{\xi}$. The incrementally solved error function is:

$$r_i(\boldsymbol{\xi}_{k+1}) = I_2(w(\boldsymbol{\xi}_k \boxplus \Delta\boldsymbol{\xi}, \boldsymbol{x}_i)) - I_1(\boldsymbol{x}_i). \tag{4.4}$$

At every iteration the function is re-linearized around the last estimate $\boldsymbol{\xi}_k$. Plugging $\boldsymbol{\theta} = \boldsymbol{\xi}$ and $\boldsymbol{\alpha} = \boldsymbol{\xi}_k$ into the formula of the linearized residual (cf. equation 2.33) results in

$$r_{\text{lin},i}(\boldsymbol{\xi}_{k+1})\big|_{\boldsymbol{\xi}=\boldsymbol{\xi}_k} = r_i(\boldsymbol{\xi}_k) + J_i(\boldsymbol{\xi}_k)\Delta\boldsymbol{\xi}. \tag{4.5}$$

The residual $r_i(\boldsymbol{\xi}_k)$ is given as

$$r_i(\boldsymbol{\xi}_k) = I_2(w(\boldsymbol{\xi}_k, \boldsymbol{x}_i)) - I_1(\boldsymbol{x}_i). \tag{4.6}$$

The 1×6 Jacobian $J_i(\boldsymbol{\xi}_k)$ contains the partial derivatives of the i^{th} pixel with respect to $\boldsymbol{\xi}$ evaluated at $\boldsymbol{\xi} = \boldsymbol{\xi}_k$. Applying the chain rule, it is calculated from $I_2(w(\boldsymbol{\xi}, \boldsymbol{x}_i))$. Therefore, it can be decomposed into a product of Jacobians, i.e.

$$\begin{aligned}
J_i(\boldsymbol{\xi}_k) &= J_I J_w && (4.7)\\
&= J_I J_\pi J_g J_G && (4.8)\\
&= \left.\frac{\partial I_2(\boldsymbol{x})}{\partial \pi}\right|_{\boldsymbol{x}=\pi(g(G(\boldsymbol{\xi}_k),\boldsymbol{p}_i))} \cdot \left.\frac{\partial \pi(\boldsymbol{p})}{\partial g}\right|_{\boldsymbol{p}=g(G(\boldsymbol{\xi}_k),\boldsymbol{p}_i)} \cdot \\
&\quad \left.\frac{\partial g(G,\boldsymbol{p})}{\partial G}\right|_{G=G(\boldsymbol{\xi}_k),\ \boldsymbol{p}=\boldsymbol{p}_i} \cdot \left.\frac{\partial G(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\boldsymbol{\xi}_k} && (4.9)
\end{aligned}$$

where $\boldsymbol{p}_i = (x, y, z)^T = \pi^{-1}(\boldsymbol{x}_i, Z_1(\boldsymbol{x}_i))$ is the 3D point corresponding to the i^{th} pixel. J_I is the 1×2 matrix of the image derivatives in x and y direction. J_π is the 2×3 matrix of derivatives of the projection function with respect to the point coordinates. J_g is the 3×12 Jacobian of the rigid body transformation with respect to its twelve parameters. J_G is the 12×6 Jacobian matrix of the exponential map (cf. equation 2.16) with respect to $\boldsymbol{\xi}$.

The definitions of the four Jacobians are

$$J_I = \left.\frac{\partial I_2(\boldsymbol{x})}{\partial \pi}\right|_{\boldsymbol{x}=\pi(g(G(\boldsymbol{\xi}_k),\boldsymbol{p}_i))} = \begin{pmatrix} \nabla I_{2,x} & \nabla I_{2,y} \end{pmatrix}, \tag{4.10}$$

$$J_\pi = \left.\frac{\partial \pi(\boldsymbol{p})}{\partial g}\right|_{\boldsymbol{p}=g(G(\boldsymbol{\xi}_k),\boldsymbol{p}_i)} = \begin{pmatrix} f_x \frac{1}{z'} & 0 & -f_x \frac{x'}{z'^2} \\ 0 & f_y \frac{1}{z'} & -f_y \frac{y'}{z'^2} \end{pmatrix}, \tag{4.11}$$

$$J_g = \left.\frac{\partial g(G,\boldsymbol{p})}{\partial G}\right|_{G=G(\boldsymbol{\xi}_k),\ \boldsymbol{p}=\boldsymbol{p}_i} = \begin{pmatrix} x & 0 & 0 & y & 0 & 0 & z & 0 & 0 & 1 & 0 & 0 \\ 0 & x & 0 & 0 & y & 0 & 0 & z & 0 & 0 & 1 & 0 \\ 0 & 0 & x & 0 & 0 & y & 0 & 0 & z & 0 & 0 & 1 \end{pmatrix}, \tag{4.12}$$

$$J_G = \left.\frac{\partial G(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\boldsymbol{\xi}_k} = \begin{pmatrix} 0 & 0 & 0 & 0 & r_{31} & -r_{21} \\ 0 & 0 & 0 & -r_{31} & 0 & r_{11} \\ 0 & 0 & 0 & r_{21} & -r_{11} & 0 \\ 0 & 0 & 0 & 0 & r_{32} & -r_{22} \\ 0 & 0 & 0 & -r_{32} & 0 & r_{12} \\ 0 & 0 & 0 & r_{22} & -r_{12} & 0 \\ 0 & 0 & 0 & 0 & r_{33} & -r_{23} \\ 0 & 0 & 0 & -r_{33} & 0 & r_{13} \\ 0 & 0 & 0 & r_{23} & -r_{13} & 0 \\ 1 & 0 & 0 & 0 & t_z & -t_y \\ 0 & 1 & 0 & -t_z & 0 & t_x \\ 0 & 0 & 1 & t_y & -t_x & 0 \end{pmatrix} \tag{4.13}$$

where x', y', z' are the transformed 3D coordinates of $\boldsymbol{p}_i$ (see equation 2.15). The product of the Jacobian matrices of the image and the warp function is:

$$J_I J_w = \begin{pmatrix} \nabla I_{2,x} & \nabla I_{2,y} \end{pmatrix} \begin{pmatrix} f_x \frac{1}{z'} & 0 & -f_x \frac{x'}{z'^2} & -f_x \frac{x' \cdot y'}{z'^2} & f_x (1 + \frac{x'^2}{z'^2}) & -f_x \frac{y'}{z'} \\ 0 & f_y \frac{1}{z'} & -f_y \frac{y'}{z'^2} & -f_y (1 + \frac{y'^2}{z'^2}) & f_y \frac{x' \cdot y'}{z'^2} & f_y \frac{x'}{z'} \end{pmatrix}. \tag{4.14}$$

The residual $r_i(\boldsymbol{\xi}_k)$ (see equation 4.6) and the Jacobian J_I depend on the warped location in the second image I_2 and therefore have to be recomputed at every iteration. As stated in equation 4.14 the Jacobian J_w changes at every iteration because the point $\boldsymbol{p}_i$ reconstructed from the first depth image Z_1 is transformed with the latest estimate $\boldsymbol{\xi}_k$.

4.2.2. Forward Compositional Algorithm

The Forward Compositional Algorithm solves the image alignment problem iteratively, as well. The main difference to the Forward Additive Algorithm is, that it chains the warping of the pixel coordinates instead of concatenating the estimates. To formalize this, instead of computing

$$r_i(\boldsymbol{\xi}_{k+1}) = I_2(w(\boldsymbol{\xi}_k \boxplus \Delta\boldsymbol{\xi}, \boldsymbol{x}_i)) - I_1(\boldsymbol{x}_i) \tag{4.15}$$

in every iteration, the following modified formulation is used

$$r_i(\boldsymbol{\xi}_{k+1}) = I_2(w(\boldsymbol{\xi}_k, w(\Delta\boldsymbol{\xi}, \boldsymbol{x}_i))) - I_1(\boldsymbol{x}_i). \tag{4.16}$$

In every iteration, this equation is linearized around $\Delta\boldsymbol{\xi} = \mathbf{0}$, which results in

$$r_{\text{lin},i}(\boldsymbol{\xi}_{k+1})\big|_{\Delta\boldsymbol{\xi}=\mathbf{0}} = r_i(\mathbf{0}) + J_i(\mathbf{0})\Delta\boldsymbol{\xi} \tag{4.17}$$

with the residual defined as

$$r_i(\mathbf{0}) = I_2(w(\boldsymbol{\xi}_k, w(\mathbf{0}, \boldsymbol{x}_i))) - I_1(\boldsymbol{x}_i) = I_2(w(\boldsymbol{\xi}_k, \boldsymbol{x}_i)) - I_1(\boldsymbol{x}_i) \tag{4.18}$$

and the Jacobian being

$$\begin{aligned} J_i(\mathbf{0}) &= J_I J_w && (4.19)\\ &= J_I J_\pi J_g J_G && (4.20)\\ &= \left.\frac{\partial I_2(w(\boldsymbol{\xi}_k, \boldsymbol{x}))}{\partial \pi}\right|_{\boldsymbol{x}=\pi(g(G(\mathbf{0}),\boldsymbol{p}_i))=\boldsymbol{x}_i} \cdot \left.\frac{\partial \pi(\boldsymbol{p})}{\partial g}\right|_{\boldsymbol{p}=g(G(\mathbf{0}),\boldsymbol{p}_i)=\boldsymbol{p}_i} \cdot \\ &\quad \left.\frac{\partial g(G, \boldsymbol{p})}{\partial G}\right|_{G=G(\mathbf{0})=I,\ \boldsymbol{p}=\boldsymbol{p}_i} \cdot \left.\frac{\partial G(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\mathbf{0}}. && (4.21) \end{aligned}$$

For linearization around $\Delta\boldsymbol{\xi} = \mathbf{0}$ the evaluation points of the Jacobians simplify, because $G(\mathbf{0})$ is the identity transform I.

Therefore, the definitions of the Jacobians become

$$J_I = \left.\frac{\partial I_2(w(\boldsymbol{\xi}_k, \boldsymbol{x}))}{\partial \pi}\right|_{\boldsymbol{x}=\pi(g(G(\mathbf{0}),\boldsymbol{p}_i))=\boldsymbol{x}_i} = \begin{pmatrix} \nabla I_{2,x} & \nabla I_{2,y} \end{pmatrix}, \tag{4.22}$$

$$J_\pi = \left.\frac{\partial \pi(\boldsymbol{p})}{\partial g}\right|_{\boldsymbol{p}=g(G(\mathbf{0}),\boldsymbol{p}_i)=\boldsymbol{p}_i} = \begin{pmatrix} f_x\frac{1}{z} & 0 & -f_x\frac{x}{z^2} \\ 0 & f_y\frac{1}{z} & -f_y\frac{y}{z^2} \end{pmatrix}, \tag{4.23}$$

$$J_g = \left.\frac{\partial g(G,\boldsymbol{p})}{\partial G}\right|_{G=G(\mathbf{0})=I,\ \boldsymbol{p}=\boldsymbol{p}_i} = \begin{pmatrix} x & 0 & 0 & y & 0 & 0 & z & 0 & 0 & 1 & 0 & 0 \\ 0 & x & 0 & 0 & y & 0 & 0 & z & 0 & 0 & 1 & 0 \\ 0 & 0 & x & 0 & 0 & y & 0 & 0 & z & 0 & 0 & 1 \end{pmatrix}, \tag{4.24}$$

$$J_G = \left.\frac{\partial G(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\mathbf{0}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{4.25}$$

The product of the Jacobian matrices of the second image and the warp function turns into

$$J_I J_w = \begin{pmatrix} \nabla I_{2,x} & \nabla I_{2,y} \end{pmatrix} \begin{pmatrix} f_x\frac{1}{z} & 0 & -f_x\frac{x}{z^2} & -f_x\frac{x\cdot y}{z^2} & f_x(1+\frac{x^2}{z^2}) & -f_x\frac{y}{z} \\ 0 & f_y\frac{1}{z} & -f_y\frac{y}{z^2} & -f_y(1+\frac{y^2}{z^2}) & f_y\frac{x\cdot y}{z^2} & f_y\frac{x}{z} \end{pmatrix}. \tag{4.26}$$

From this equation its apparent, that J_w is constant because it only depends on the point coordinates $\boldsymbol{p}_i$ reconstructed from the first depth map Z_1. Therefore, the Jacobian J_w does not need to be recalculated at every iteration, in contrast to the Forward Additive Algorithm.

For every iteration the warped second image $I_2(w(\boldsymbol{\xi}_k, \boldsymbol{x}))$ and its derivative in x and y direction (for J_I see equation 4.22) have to be calculated. In [54] the Jacobian J_I is approximated by the gradients computed in the first image I_1 instead of the warped second image.

4.3. Robustification

For stabilization and position control robust motion estimates are mandatory and gross errors should be avoided. The term *robust* is used to describe that the estimation methods are not influenced by outliers present in the sensor data. Outliers are all errors, which cannot be explained by the model underlying the estimation method. The main problems with the approach of Steinbrücker et al. [47] are sensitivity to outliers in the images and occasional jumps in the estimated trajectory. In the following, the reasons for outliers and jumps are discussed and the implemented countermeasures are detailed.

The outliers have different causes, e.g. occlusions, dynamic objects moving through the scene, illumination changes, or sensor noise. Especially dynamic objects, like persons, moving differently than the camera *induce* a motion different from the true camera motion leading to wrong estimates. Outliers are malicious in least squares estimation, because they cause large residuals which have a huge influence on the estimate due to the quadratic error term.

In general, there are two possibilities to deal with outliers. The first option is to model the effects causing outliers and estimate their parameters, as well. For example, Black et al. [6] model several effects causing outliers in images and simultaneously estimates the parameters for all components. The second possibility is to suppress outliers. As Black et al. still include a generic model for all outliers not represented by any of the specialized models and only the estimate of the camera motion is of interest, the option suppressing outliers is chosen.

From a Bayesian perspective, the robustness can be increased by choosing different probability distributions for the residuals $p(\boldsymbol{r} \mid \boldsymbol{\xi})$ and the parameters $p(\boldsymbol{\xi})$ (cf. section 2.5.2). Alternate models for the residual distribution are incorporated through different weight functions in weighted least squares (cf. section 2.5.3). The distribution $p(\boldsymbol{\xi})$ models a prior on the motion and can be integrated as described in section 2.5.2.

In the following, the extension to a weighted least squares approach and the different choices for the weight functions are discussed. Afterwards the options for priors on the motion parameters are presented.

4.3.1. Iteratively Re-Weighted Least Squares

In section 2.5.3 the weighted least squares method was introduced as a mean to robustify least squares problems against outliers. Robust error functions as introduced in robust statistics literature [25, 19] have previously been used in computer vision [53, 48, 9, 30, 31].

Tykkälä et al. use weighting based on robust statistics. They choose the Tukey function for weight calculation. The Tukey function requires a scale estimate of the error distribution, it is robustly computed using the Median Absolute Deviation (cf. equation 2.50). The Tukey weight function $\omega_{\text{Tukey}}(x)$ is defined as:

$$\omega_{\text{Tukey}}(x) = \begin{cases} \left(1 - \frac{x^2}{b^2}\right)^2 & |x| \leq b \\ 0 & \text{else} \end{cases} \tag{4.27}$$

where the hyper parameter b is chosen as 4.6851 to achieve 95% efficiency if the outlier-free distribution is assumed to be Gaussian [60]. The weight w_{ii} for the i^{th} residual r_i in the weight matrix W (cf. section 2.5.3) is computed as:

$$w_{ii} = \omega_{\text{Tukey}}\left(\frac{r_i}{\sigma_{\text{MAD}}}\right). \tag{4.28}$$

Figure 4.4a shows a plot of the Tukey weight function. It depicts, that the weights for large residuals decrease to zero. Therefore, outliers are completely suppressed. While the Tukey weights successfully suppress outliers, like moving persons, the tracking performance is degraded as well (see chapter 5).

Other researchers choose the Huber function [41, 57]. It gives small residuals quadratic and large residuals only linear influence. In contrast to the Tukey function it is convex and therefore does not introduce new local minima to the optimization. The Huber function is defined as:

$$\omega_{\text{Huber}}(x) = \begin{cases} 1 & |x| \leq k \\ \frac{k}{|x|} & \text{else} \end{cases} \tag{4.29}$$

with $k = 1.345$ for a Gaussian distribution [60]. Figure 4.4a depicts a corresponding plot. The Huber based weights do not degrade the tracking performance as the Tukey weights do, but they are worse in supressing outliers. For a detailed evaluation refer to chapter 5.

As the two different weight functions do not provide satisfying results according to preliminary experiments a different approach is selected. As shown in section 2.5.3 the weight function is generically defined as $\omega(x) = \phi(x)/x$ where $\phi(x)$ is the derivative of $\rho(x)$. The function $\rho(x)$ is, in fact, the negative logarithm of the residual distribution $p(\boldsymbol{r} \mid \boldsymbol{\theta})$. Therefore, the weight function is directly dependent on the distribution of the residuals. Based on this insight the distribution of the residuals is empirically determined and a suitable analytical distribution is chosen. The weight function is derived from the analytical distribution.

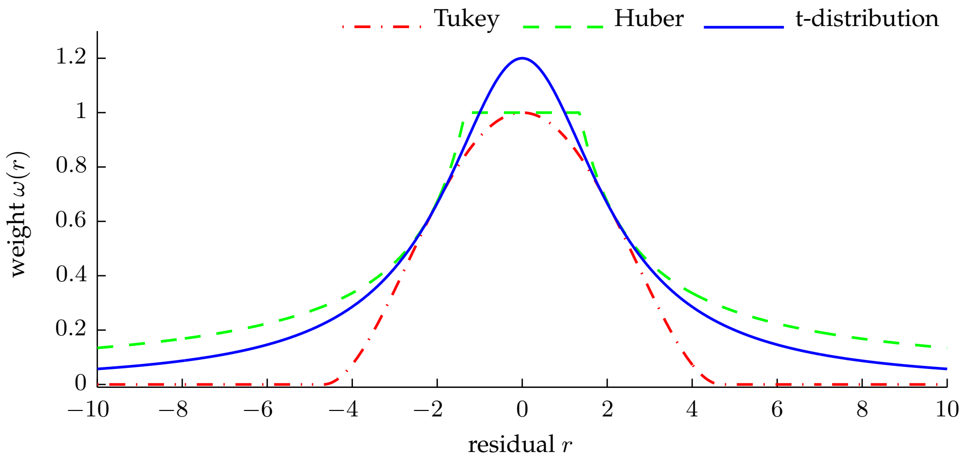
To obtain the empirical distribution of the residuals, the error between consecutive intensity images of a whole sequence is computed. The errors of each image pair are accumulated in a histogram. As every error is assumed to be drawn from the same distribution (cf. iid. assumption section 2.5), the histogram is a representative of the underlying distribution. Figure 4.5 shows several example histograms from different image sequences. To increase the sample size (every image pair provides $640 \cdot 480 = 307200$ samples), the single histograms are accumulated in one histogram for the whole image sequence. This is done for several image sequences including self-recorded sequences and sequences from the RGB-D benchmark [51]. The content of the sequences also varies from static to dynamic scenery. In figure 4.6 the accumulated histograms for three sequences (*fr1/desk*, *fr3/sitting_halfsphere*, *fr3/walking_halfsphere*) are shown.

Different distributions are fitted to the accumulated histograms. Figure 4.7 shows the histogram of the *fr2/desk* sequence with fitted normal distribution, robust normal distribution, and t-distribution. Several authors proposed the t-distribution for robust data fitting [33, 15] and it has been applied successfully in computer vision problems [16, 36, 14]. For all distributions the mean was fixed to 0. The standard deviation σ for the robust normal distribution has been calculated using the MAD estimator (cf. equation 2.50). The parameters for the t-distribution have been obtained by their maximum likelihood estimators [34]. As shown in the figure the normal distributions do not fit the histogram, while the t-distribution fits nicely.

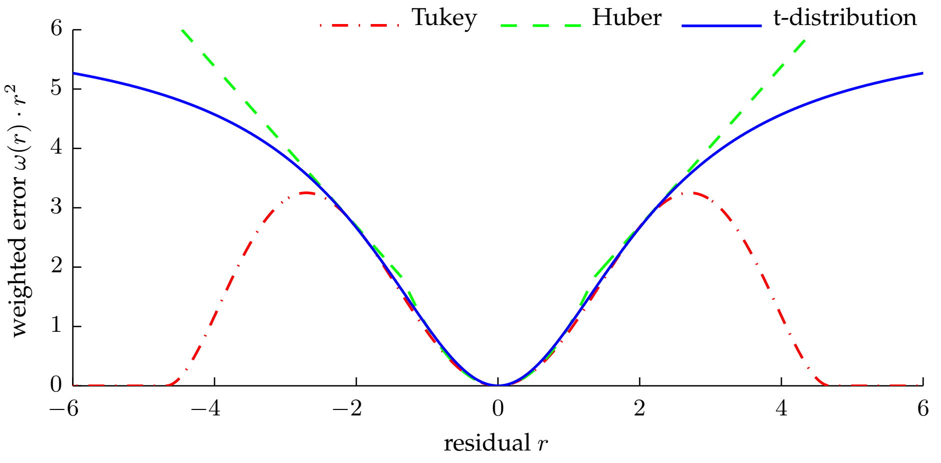
The t-distribution is defined as:

$$p(x \mid \mu, \sigma, \nu) = \frac{\Gamma(\frac{v+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\pi\nu\sigma^2}} \left(1 + \frac{1}{\nu}\frac{(x-\mu)^2}{\sigma^2}\right)^{-\frac{v+1}{2}} \tag{4.30}$$

where μ is the mean, σ is the standard deviation and ν are the degrees of freedom [5]. The gamma function is $\Gamma(x) = (x-1)!$. A plot of the probability density function of the



(a) Different weight functions $\omega(x)$: Tukey with $b = 4.685$ (red), Huber with $k = 1.345$ (green) and t-distribution with $\nu = 5$ (blue)



(b) Weighted error $\omega(r) \cdot r^2$ for different weight functions: Tukey with $b = 4.685$ (red), Huber with $k = 1.345$ (green) and t-distribution with $\nu = 5$ (blue)

Figure 4.4.: Different weight functions (a) control the influence of residual values (b) on the optimization. Large residuals stem typically from outliers and are assigned low weights.

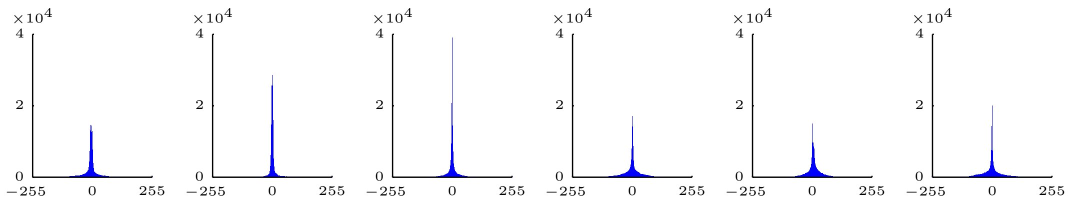
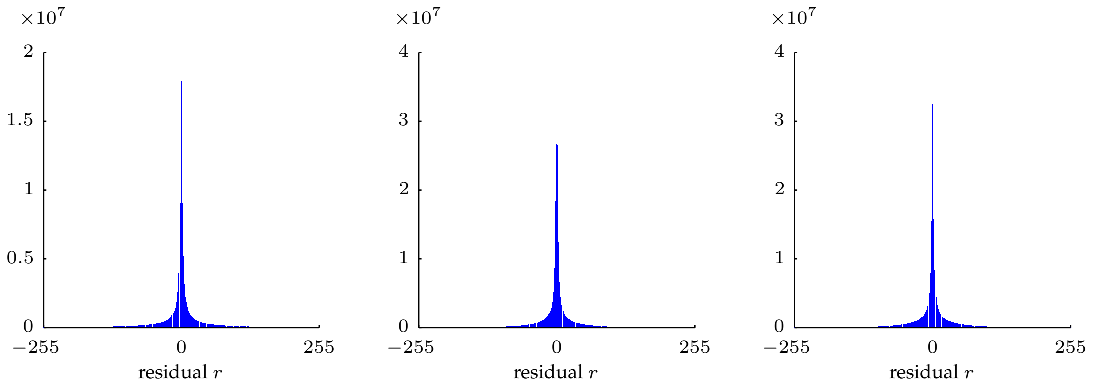


Figure 4.5.: Histograms of the residuals of different image pairs from the *fr1/desk*, *fr3/sitting* and *fr3/walking* sequences. The bin with the most elements is always 0 while the spread of the histograms varies.



(a) *fr1/desk* (b) *fr3/sitting_halfsphere* (c) *fr3/walking_halfsphere*

Figure 4.6.: Accumulated residual histograms over all image pairs of the *fr1/desk* (a), *fr3/sitting* (b) and *fr3/walking* (c) sequences. The shape of the histograms is similar and resembles the probability densitiy function of the t-distribution.

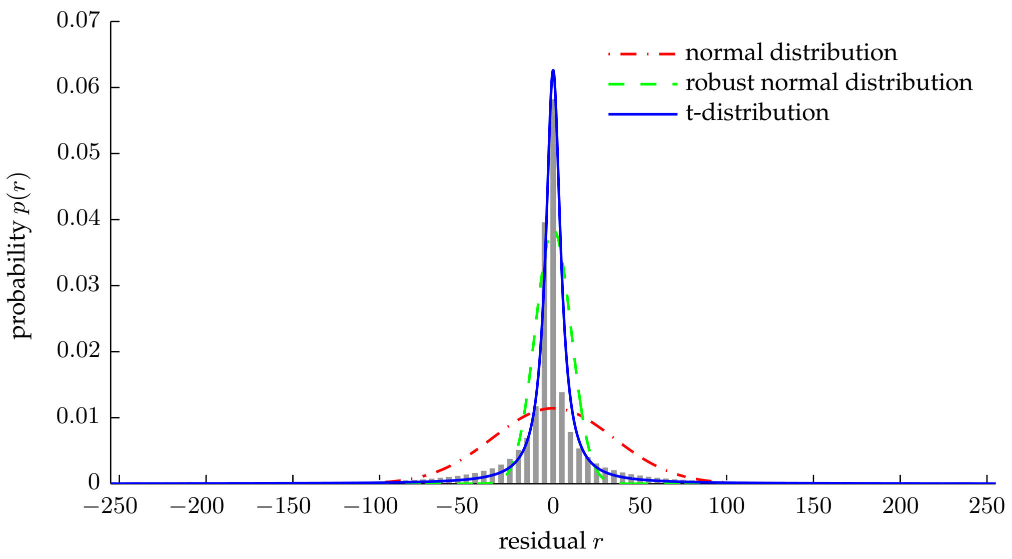


Figure 4.7.: Different probability density functions are fitted to the accumulated histogram of the *fr2/desk* sequence. The mean of all distributions is fixed to zero. The normal distribution (red) and the robust normal distribution (green) do not accuratly represent the underlying empirical distribution represented by the histogram. In contrast, the t-distribution (blue) fits nicely.

t-distribution is depicted in figure 4.7 (blue). For $\nu \rightarrow \infty$ the t-distribution approaches the normal distribution.

Bishop interprets the t-distribution as an infinite mixture of Gaussian distributions [5]. In the context of least squares optimization this is a more flexible model than the iid. assumption introduced in section 2.5.2. The residuals are still assumed to originate from independent distributions, but not from identical distributions. Without the assumption of an identical normal distribution the variance cannot be dropped in equation 2.42 and every residual is weighted with its inverse variance. Distributions with high variance are the source of outliers, which therefore get low weights. While the estimation of the variance for every residual is intractable the t-distribution provides a feasible abstraction.

The weight function resulting from the t-distribution is given as:

$$\omega_{\mathrm{t}}(x) = \frac{\nu + 1}{\nu + (\frac{x}{\sigma_{\mathrm{t}}})^2}. \tag{4.31}$$

The degrees of freedom parameter is fixed to 5 based on experiments (see chapter 5). The scale parameter is estimated in every iteration based on the current residuals using its maximum likelihood estimator

$$\sigma^2_{\mathrm{t},k+1} = \frac{1}{n} \sum_{i}^{n} \frac{\nu + 1}{\nu + r_i^2 / \sigma^2_{\mathrm{t},k}} r_i^2 \tag{4.32}$$

which has to be solved iteratively. The mean μ is dropped in equations 4.31 and 4.32, because it is assumed to be 0. A plot of the t-distribution weight function is shown in figure 4.4a.

In figure 4.8a a scene with a hand moving in the opposite direction of the camera is shown. As displayed in the residual image (figure 4.8b) calculated after convergence, the hand causes outlier pixels (white). The three images in figure 4.8 (e,c,d) show the weights based on the different weight functions. Darker pixels indicate lower weights with a zero weight being black. The hand generating outliers gets downweighted by all three functions. The book covers and the shirt clearly show the lower or zero weights from the Tukey function.

Chapter 5 gives a detailed evaluation of the performance of the different weight functions. Based on these results the t-distribution based weight function was chosen to robustify the least squares problem.

4.3.2. Motion Priors

Knowledge about the camera motion can be integrated into the estimation to regularize the equation system and to provide a better initialization for the optimization. The following assumptions are valid with respect to the camera motion:

- it is not entirely random, no matter whether it is handheld or mounted on a quadrocopter, but follows a smooth trajectory,
- it has to be small, otherwise the optimization might not converge to the correct solution.

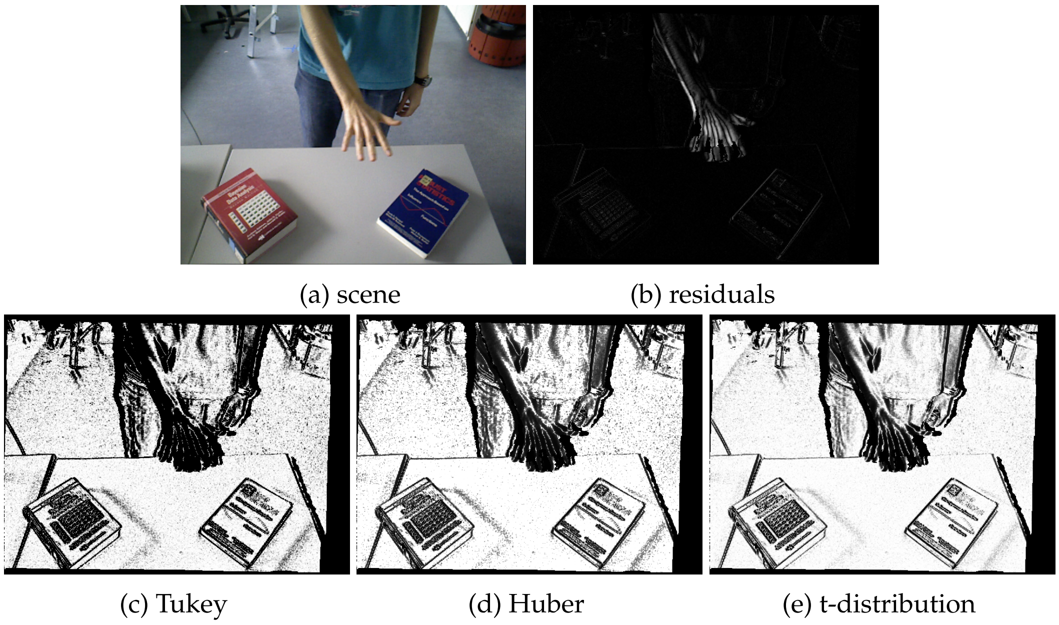


Figure 4.8.: In this experiment a hand moves in the opposite direction than the camera (a). In the residual image computed after convergence the hand still causes outliers (b). All weight functions assign small weights to the outliers caused by the moving hand. The functions vary in the weighting of inlier pixels. For example, the book covers and the shirt show that the Tukey weight function assigns in general more small weights (c), in contrast to the Huber (d) and t-distribution (e) function.

Furthermore, the quadrocopter caries additional sensors, e.g. an IMU, which also provide motion information.

Every prior gives parameters $\boldsymbol{\xi}_{\text{prior}}$. To express the smooth trajectory assumption the prior is set to the motion estimate between the last image pair, i.e. $\boldsymbol{\xi}_{\text{prior}} = \boldsymbol{\xi}_{t-1}$. For the small motion assumption $\boldsymbol{\xi}_{\text{prior}}$ is chosen to be the identity: $\boldsymbol{\xi}_{\text{prior}} = \mathbf{0}$. When using an additional sensor $\boldsymbol{\xi}_{\text{prior}}$ can be set to this motion estimate, e.g. $\boldsymbol{\xi}_{\text{prior}} = \boldsymbol{\xi}_{\text{IMU}}$.

Independent of the choice $\boldsymbol{\xi}_{\text{prior}}$ there are two possibilities to incorporate it into the optimization. The first one is to use the initial estimate to initialize the optimization, i.e. $\boldsymbol{\xi}_0 = \boldsymbol{\xi}_{\text{prior}}$. The second option is to incorporate the prior through the prior distribution $p(\boldsymbol{\xi})$ (see equation 2.38) into the equations.

The integration as initial guess can accelerate the convergence of the optimization and guide it to the correct minimum. Bad choice of $\boldsymbol{\xi}_0$ can slow down the algorithm or cause convergence to a wrong minimum.

In contrast, the integration through the prior distribution always affects the solution, because the normal equations are modified (cf. equation 2.44). The influence can be controlled through the inverse variance $\sigma_{\text{prior}}^{-2}$ or in the multivariate case through the inverse covariance matrix $\Sigma_{\text{prior}}^{-1}$.

The regularization using the identity prior is known as Tykhonov regularization [5]. A challenge when using the estimates of the IMU as prior is that precise inter-sensor calibration in terms of relative pose to each other and time synchronization is required. The effects of incorporating the different priors are discussed in chapter 5.

4.4. Implementation

Applied on board of the quadrocopter the visual odometry approach requires an efficient implementation. Furthermore, it has to be integrated with the RGB-D camera drivers, the control and the data fusion software.

The system is implemented on the Robot Operating System (ROS) middleware. The main reasons for this choice are that the driver framework for the AscTec Pelican (see section 2.1.2) is implemented in ROS, and the availability of drivers for the RGB-D camera.

Figure 4.9 depicts the overall structure of the software components and their communication paths. The *openni_driver* component integrates the ASUS Xtion PRO LIVE drivers and provides the RGB-D images to the *camera_tracker*. The *camera_tracker* implements the presented visual odometry approach. The distributed EKF framework uses the pose estimates to correct the state predictions. These predictions are calculated from the IMU data at 1 kHz on the Autopilot Board of the Pelican. Finally, the position controller uses the state estimate of the EKF, which includes the fused data from the visual odometry and the IMU, to stabilize the quadrocopter.

The main contribution of this thesis is the *camera_tracker* component. It implements the camera motion estimation from an RGB-D image pair using the theoretical foundations described in section 4.2 and 4.3. The pseudo code of the core function is given in algorithm 1. It applies a coarse to fine scheme, also used in [47], to estimate the camera motion. Starting with a low resolution version of the images allows to compensate for larger motions and accelerates computation. At every level l multiple iterations are performed. In every iteration k the Jacobian matrix J, the weighting matrix W and the residuals $\boldsymbol{r}$ are computed.

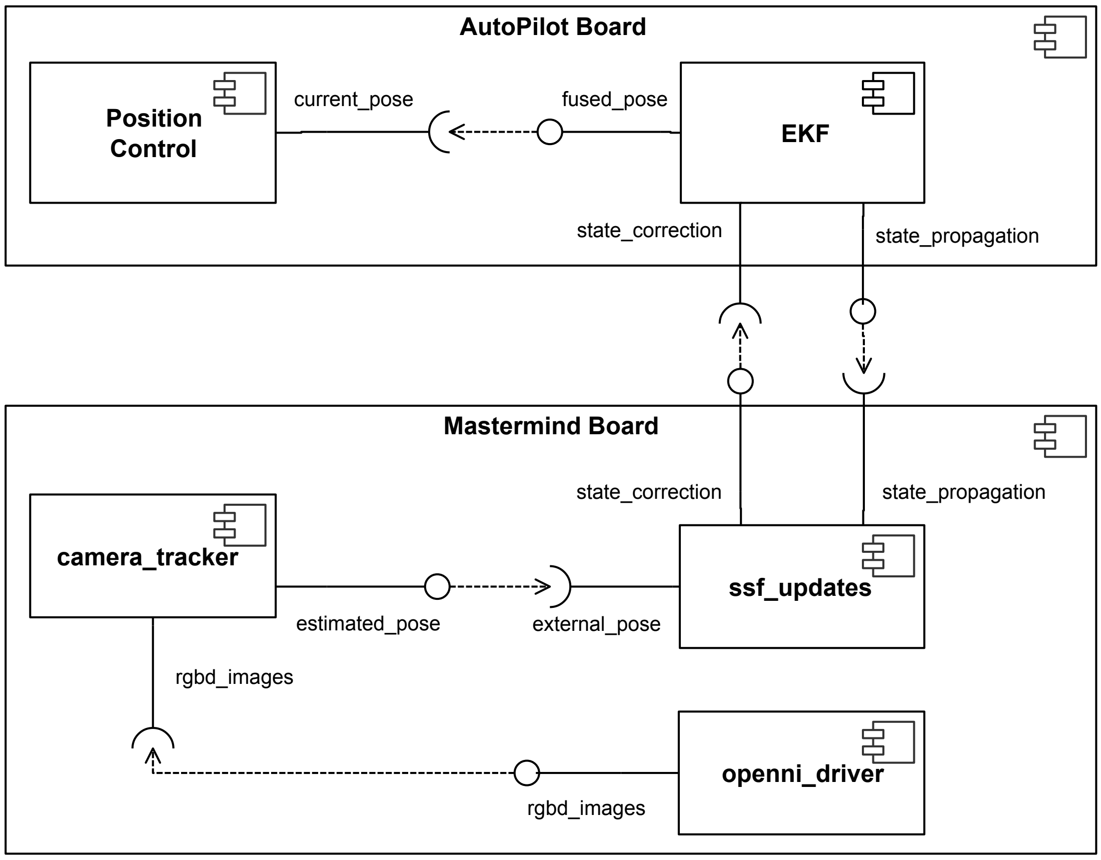


Figure 4.9.: The software components and their integration on the Pelican quadrocopter. The *openni_driver* provides the RGB-D images. The *camera_tracker* implements the presented visual odometry approach. The distributed EKF uses the estimated pose from the visual odometry to correct its state estimate.

In case the error e decreased with the last increment, it is incorporated into the current estimate. Afterwards, the new increment is calculated by solving the normal equations. Iterating at the current level is stopped when the error increases, the error decrement is below a user defined threshold, or the maximum number of iterations is exceeded.

Algorithm 1 Camera motion estimation from an RGB-D image pair

```
1:  function EstimateCameraMotion($I_{1,\text{pyr}}, Z_{1,\text{pyr}}, I_{2,\text{pyr}}, \boldsymbol{\xi}_{\text{initial}}$)
2:      $\boldsymbol{\xi} \leftarrow \boldsymbol{\xi}_{\text{initial}}$
3:      $\Delta\boldsymbol{\xi} \leftarrow \mathbf{0}$

4:      for $l$ = CoarsestLevel $\rightarrow$ FinestLevel do
5:          $e \leftarrow 0$
6:          $e_{\text{last}} \leftarrow \text{inf}$
7:          $k \leftarrow 0$

8:          while $(e_{\text{last}} - e) > \epsilon$ and $k_{\max} > k$ do
9:              $J, W, \boldsymbol{r} \leftarrow$ ComputeJacobianAndError($I_{1,\text{pyr}}(l), Z_{1,\text{pyr}}(l), I_{2,\text{pyr}}(l), \boldsymbol{\xi} \boxplus \Delta\boldsymbol{\xi}$)

10:             $e_{\text{last}} \leftarrow e$
11:             $e \leftarrow \frac{1}{n}\boldsymbol{r}^T W \boldsymbol{r}$

12:             if $e > e_{\text{last}}$ then                     ▷ ensure error decreased
13:                 $\Delta\boldsymbol{\xi} \leftarrow \mathbf{0}$
14:             else
15:                 $\boldsymbol{\xi} \leftarrow \boldsymbol{\xi} \boxplus \Delta\boldsymbol{\xi}$                   ▷ incorporate increment
16:                 $\Delta\boldsymbol{\xi} \leftarrow (J^T W J)^{-1} J^T W \boldsymbol{r}$       ▷ solve normal equations
17:             end if

18:             $k \leftarrow k + 1$
19:         end while
20:     end for

21:     return $\boldsymbol{\xi}$
22: end function
```

The integration of motion priors is not shown in algorithm 1. It requires modification of line 16 according to equation 2.44. The most critical part in algorithm 1 is the computation of the Jacobians, the weights and the residuals (see line 9). It implements the Forward Compositional Algorithm (see section 4.2.2) and is shown in algorithm 2. The scale is estimated (line 4) either using the median absolute deviation or the maximum likelihood estimator of the t-distribution depending on the weight function. The weights are calculated from the normalized residuals (line 5) using one of the weight functions introduced in section 4.3. The derivatives of the warped second image I_2^* in x and y direction are computed using central differences.

As the *ComputeJacobianAndError* function is called in every iteration and iterates over all image pixels it is decisive for the performance of the whole implementation. Therefore,

Algorithm 2 Computation of the Jacobian, weights and residual for all pixels according to the Forward Compositional Algorithm (see section 4.2.2)

```
1:  function ComputeJacobianAndError($I_1, Z_1, I_2, \boldsymbol{\xi}$)
2:      $I_2^* \leftarrow I_2(w(\boldsymbol{x}, \boldsymbol{\xi}))$
3:      $\boldsymbol{r} \leftarrow I_2^* - I_1$
4:      $\sigma \leftarrow$ ComputeScale($\boldsymbol{r}$)
5:      $W \leftarrow$ ComputeWeights($\boldsymbol{r}/\sigma$)
6:      $\nabla I_{2,x}^*, \nabla I_{2,y}^* \leftarrow$ ComputeDerivatives($I_2^*$)

7:      for $i = 0 \rightarrow n$ do
8:          $J(i) \leftarrow$ ComputeJacobian($\pi^{-1}(\boldsymbol{x}(i), Z_1(i)), \nabla I_{2,x}^*(i), \nabla I_{2,y}^*(i)$)
9:      end for

10:     return $J, W, \boldsymbol{r}$
11: end function
```

several optimizations have been applied. Most important is the use of Streaming SIMD Extensions (SSE) instructions, where SIMD abbreviates Single Instruction Multiple Data. They allow to apply the same operation to multiple values at the same time. This is beneficial if the same calculations have to be performed on many values as is the case in the presented algorithm. SSE instructions are available on all recent Intel CPUs, including the Intel Core2Duo on the AscTec Mastermind board (cf. section 2.1.2). The caching of J_w as suggested in section 4.2.2 has no significant impact on the performance. A further optimization would be to parallelize the computations on multiple CPU cores. However, this is not subject of this thesis.

5. Evaluation

This chapter describes the experiments evaluating the performance of the proposed approach and presents the results. These experiments consist of three parts. The first part evaluates the performance of the visual odometry algorithm on different simulated and RGB-D benchmark datasets. Second, the integration on the quadrocopter and the position control using the presented approach are evaluated. The last part evaluates the influence of the different parameters and discusses based on these findings the optimal parameter values for the application scenario.

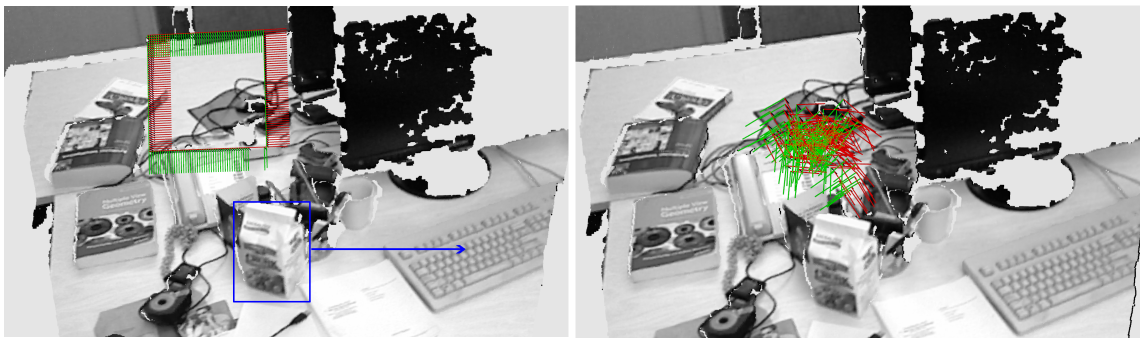
5.1. Performance Comparison

The following experiments are conducted to evaluate the performance of the proposed algorithm in comparison to the implementation of Steinbrücker et al., and to implementations using the Tukey and the Huber function for weight calculation. There are two groups of experiments. The first group uses synthetic datasets with perfect ground truth and ideal RGB-D images. The second group evaluates the performance on the RGB-D benchmark [51].

5.1.1. Synthetic Datasets

The synthetic datasets are generated from an RGB-D image of the *fr1/desk* sequence by warping it along a predefined trajectory. Two different trajectories are used. The first simulates motion along a square with edge length of 10 cm with a displacement of 2 mm between two images. This trajectory simulates a camera moving at 0.06 m/s with a framerate of 30 Hz. The second trajectory is random. It translates the camera along the x and y axis of the camera plane and rotates around the z axis of the camera. The translations are sampled from a uniform distribution in the range of $\pm 1\,\mathrm{cm}$. The angle of rotation is uniformly drawn from $\pm 5°$. This virtual camera has an average speed of 0.3 m/s. Figure 5.1 visualizes the two trajectories and the point cloud of the scene. For each of the two trajectories an additional dataset is generated by moving a small patch of the scene independent from the camera. This moving patch represents a motion, which causes outliers. Figure 5.1a depicts the patch and its motion vector.

On each of the four datasets the original implementation provided by Steinbrücker et al. is executed as baseline for comparison. Afterwards the four different weighting variants (no weighting, Tukey, Huber, t-distribution) are run each with two sets of parameters. The first parameter set uses the realtime parameters determined in section 5.3. The second parameter set is tuned for maximum accuracy. Therefore, it uses a precision of $\epsilon = 10^{-12}$, a large value for the maximum number of iterations k_{max}, and all resolutions up to 640×480 pixels. In total there are nine evaluation runs per dataset.



(a) synthetic datatset along square trajectory (b) synthetic datatset along random trajectory

Figure 5.1.: Synthetic datasets with different trajectories created from an RGB-D image of the *fr1/desk* sequence. The marked patch in (a) is moved independently of the virtual camera to simulate outliers.

Tables 5.1 to 5.4 list the results for each dataset. The first column states the variant of the algorithm — *reference* is the implementation of Steinbrücker et al. The second column indicates the parameter set, either *realtime* or *precision*. The third column contains the RMSE of the translational drift per second. The fourth column depicts the relative improvement compared to the *reference* implementation, which is computed as

$$\text{improvement}(\text{RMSE}) = \frac{\text{RMSE}_{\text{reference}} - \text{RMSE}}{\text{RMSE}_{\text{reference}}} \cdot 100\,\%. \tag{5.1}$$

The last column shows the average runtime per RGB-D image pair in seconds.

For the square trajectory on a static scene all variants show an improvement over the reference implementation (see table 5.1). With realtime parameters there is no significant difference in performance. For the precision parameters all variants, except the Huber weighting, improve around 10 %, but at cost of much higher runtimes. The t-distribution weighting performs best for both parameter sets. The performance degradation of the Huber weights with the high accuracy parameters stems from convergence to wrong solutions. The error seems to keep decreasing, therefore the optimization does not stop, and a large number of iterations is performed as can be seen from the high runtime. Since the high accuracy parameter set has little relevance for the application on the quadrocopter, this issue is not subject to further investigations.

The experiment on the square trajectory with a moving object shows significant performance improvements for the weighted variants in contrast to ones without weighting (see table 5.2). The unweighted version is worse than the reference implementation. This might be a result of more iterations leading to convergence to a wrong estimate, whereas the reference implementation performs fewer iterations. The performance gain provided by the precision parameters is around 50 % for the Tukey and t-distribution weighting function. Again the t-distribution weights perform best.

The experiments for the random motion dataset with faster motion exhibit a decrease in the performance of the reference implementation by factor two to three compared to the slower square trajectory sequences. Without outliers the Huber and t-distribution weights perform equally well for both parameter sets (see table 5.3). For the Tukey weights the drift

Table 5.1.: RMSE of translational drift for the synthetic square motion sequence for different weight functions and parameter sets. Robust weighting with the Huber function or t-distribution improves the performance even in the absence of gross outliers. The improvement for the precision parameter set is small compared to the increase in runtime.

Method	Parameter Set	RMSE [m/s]	Improvement	∅Runtime [s]
reference		0.0344	0.0 %	0.035
no weights	realtime	0.0151	56.1 %	0.023
Tukey weights	realtime	0.0163	52.6 %	0.040
Huber weights	realtime	0.0128	62.8 %	0.019
t-distribution weights	realtime	**0.0110**	68.0 %	0.030
no weights	precision	0.0079	77.0 %	0.126
Tukey weights	precision	0.0133	61.3 %	0.196
Huber weights	precision	0.0238	30.8 %	1.484
t-distribution weights	precision	**0.0078**	77.3 %	0.305

Table 5.2.: RMSE of translational drift for the synthetic square motion sequence with outliers for different weight functions and parameter sets. The t-distribution weights clearly outperform the Tukey and Huber weights in presence of outliers.

Method	Parameter Set	RMSE [m/s]	Improvement	∅Runtime [s]
reference		0.0377	0.0 %	0.037
no weights	realtime	0.0427	-13.3 %	0.022
Tukey weights	realtime	0.0306	18.8 %	0.035
Huber weights	realtime	0.0255	32.4 %	0.020
t-distribution weights	realtime	**0.0193**	48.8 %	0.034
no weights	precision	0.0401	-6.4 %	0.115
Tukey weights	precision	0.0132	65.0 %	0.345
Huber weights	precision	0.0254	32.6 %	1.312
t-distribution weights	precision	**0.0082**	78.2 %	0.510

Table 5.3.: RMSE of translational drift for the synthetic random motion sequence for different weight functions and parameter sets. On this dataset with comparably fast motion but no outliers the improvement due to use of the Huber and t-distribution weights is significant.

Method	Parameter Set	RMSE [m/s]	Improvement	∅Runtime [s]
reference		0.0751	0.0 %	0.038
no weights	realtime	0.0223	70.0 %	0.032
Tukey weights	realtime	0.0497	33.8 %	0.050
Huber weights	realtime	**0.0134**	82.2 %	0.025
t-distribution weights	realtime	0.0142	81.1 %	0.043
no weights	precision	0.0145	80.7 %	0.115
Tukey weights	precision	0.0279	62.8 %	0.230
Huber weights	precision	0.0125	83.4 %	1.287
t-distribution weights	precision	**0.0124**	83.5 %	0.405

increases compared to the unweighted case. In the presence of outliers the t-distribution and Tukey weights perform better than the unweighted and Huber variants. The drift for the t-distribution estimates is still the smallest.

This set of experiments proves that the implementation developed in this thesis is superior to the reference implementation of Steinbrücker et al. Furthermore, it shows that robust weighting is beneficial in the presence of outliers. As well, the experiments confirm the claims from section 4.3, that the Tukey weights degrade the tracking performance when there are few outliers and that the Huber weights are less resistant to outliers. Finally, they underpin the suitability and good performance of the t-distribution based weights.

5.1.2. RGB-D Benchmark Datasets

A second set of experiments assesses the performance on the RGB-D benchmark datasets. From the different categories of datasets the following are selected: *Testing and Debugging*, *Handheld SLAM*, *Robot SLAM* and *Dynamic Objects*. They reflect the typical scene contents and application environments. As for the synthetic datasets the reference implementation of Steinbrücker et al. and the variants with different robust weighting schemes and realtime parameters are evaluated. One additional variant uses the t-distribution weight function and the temporal prior.

The tables in Appendix A present a detailed overview of the drift for all datasets. In the following the results for each category are summarized.

On the datasets in the *Testing and Debugging* category all variants except the Tukey weights perform equally well. They achieve an improvement of 40 % with respect to the reference implementation. The Huber weights exhibit 1 mm/s less drift than the t-distribution weights. On these simple datasets the proposed robust visual odometry has a drift around 2.4 cm/s.

The datasets in the *Handheld SLAM* category provide similar results. The t-distribution and Huber weights perform equally good, again. For the fast *fr1/desk*, *fr1/desk2* and *fr1/room*

Table 5.4.: RMSE of translational drift for the synthetic square motion sequence with outliers for different weight functions and parameter sets. On this sequence with fast camera motion and outliers the t-distribution weights are superior to the other variants.

Method	Parameter Set	RMSE [m/s]	Improvement	∅Runtime [s]
reference		0.1019	0.0 %	0.035
no weights	realtime	0.0650	36.2 %	0.030
Tukey weights	realtime	0.0382	62.5 %	0.056
Huber weights	realtime	0.0572	43.9 %	0.024
t-distribution weights	realtime	**0.0296**	71.0 %	0.047
no weights	precision	0.0502	50.7 %	0.130
Tukey weights	precision	0.0270	73.5 %	0.279
Huber weights	precision	0.0457	55.2 %	0.935
t-distribution weights	precision	**0.0133**	87.0 %	0.508

these weighted variants cause several centimeters per second less drift than the unweighted one. For the rather slow and feature rich *fr3/long_office_household* sequence the reference implementation provides the best result. The average drift per second of the t-distribution weights is 4.2 cm/s and is slightly better than the one of the Huber weights and 1 cm/s better than the unweighted version.

The *Robot SLAM* category contains datasets where all variants of the approach except the one with temporal prior have a drift of more than 20 cm/s. The configuration with temporal prior still exhibits a comparably high drift of 8 cm/s. One reason is that the video shows a lot of sudden vertical shifts when the robot rolls over cables lying on the floor. Another observation is that a part of the video is missing in the *fr2/pioneer_slam2* dataset.

The last category is *Dynamic Objects*. It contains datasets where persons move through the scene with varying amount of motion and the camera follows different trajectories. The movements of the person cause outliers. Therefore, these datasets are suited to test the robustness of the approach. There are two groups of datasets. One where persons are sitting at a table, talking and moving slightly. The second group contains persons walking through the scene and obstructing large parts of the field of view. The *sitting* datasets confirm that the t-distribution and Huber weighting work equally good. The version with temporal prior has the least drift, on average. For the *sitting* datasets with rotational motion (rpy) disabled weighting performs better. This indicates that weighting suppresses rotational motion estimation, especially. The temporal prior compensates this issue. Averaged over all *sitting* datasets the drift is in the order of 3–4 cm/s. For the *walking* sequences all variants of the approach fail. Even the best version with the Tukey weight function has an average drift of 26 cm/s. Accordingly, scenes with that many outliers remain an open challenge for future research.

This set of experiments confirms the results of the previous section on realistic sequences. However, the Huber weight function is on par with the t-distribution weight function in these experiments. The implementation developed in this thesis always performs better or equally good as the previous one provided by Steinbrücker et al.

5.2. Flight Experiments

The next set of experiments evaluates the performance of the visual odometry approach on the quadrocopter. As the previous experiments show the drift is at least 2–4 cm/s theoretically accumulating to 1.2–2.4 m/min. This renders flight for several minutes or flying along trajectories infeasible. Thus, the drift while hovering at a fixed position is measured.

The experimental setup is as follows: The quadrocopter is equipped with the sensors and computer boards as described in section 2.1.2. The grayscale camera tracks external markers to obtain the ground truth trajectory and start the quadrocopter. Starting with the external tracking system is necessary, because the Pelican tends to tilt due to unleveled weight distribution. This tilt causes the RGB-D camera mounted on the lower part of the quadrocopter to face the floor. Therefore, no depth can be measured anymore and motion estimation becomes impossible. Once the quadrocopter is stable and reached its hover point the dense visual odometry is enabled and substitutes the external tracking system. Switching the tracking technique introduces further issues, for example, the visual odometry algorithm estimates only relative poses and therefore needs to use the last pose from the tracking system as initial pose. But switching requires a certain amount of time, during which the quadrocopter already moves away from this initial pose. This effect leads to a constant offset between true and estimated position.

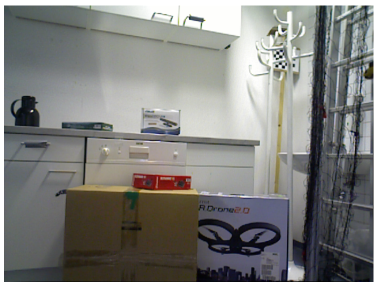
Several other factors have negative influence on the flight performance. These are:

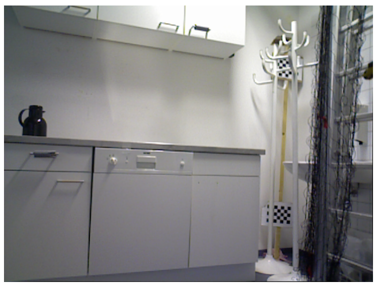
- Imprecise inter-sensor pose calibration of the RGB-D camera to the IMU. This results in incompatible measurements from the different sensors.
- Incorrect sensor delay parameter. The EKF uses the delay to synchronize the different sensors. Small, unavoidable imprecisions cause instability during flight.
- Fixed noise parameters determine how precise the visual pose estimate is. Wrong values let the EKF give too much or too little influence to the measurements on the state estimate. In practice the noise also varies between the measurements.

As the whole system is a closed loop errors in one component also effect the others. For example, wrong state estimates of the EKF cause sudden reactions of the controller to compensate for the error. These rapid attitude and position changes lead to worse estimates from the visual odometry, which again degrade the EKF state estimates. Therefore, achieving stable flight of the quadrocopter is a challenging task.

To test the approach against different environments the scene is augmented with a varying number of objects introducing further color and structure information. In total three scene setups are under investigation. Figure 5.2 shows exemplary images for each scene. The first one is very cluttered and contains many color and depth cues (see figure 5.2a). The second scene setup contains less objects and the planar, white fronts of the furniture dominate (see figure 5.2b). In the third setup mostly textureless and planar cabinet doors are visible (see figure 5.2c).

The pose estimation fails on the cabinet scene. This causes rapid drift of the quadrocopter. Therefore, no reliable data for evaluation can be recorded. For the other two scenes


(a) scene with many textural and structural cues


(b) scene with normal texture and structure

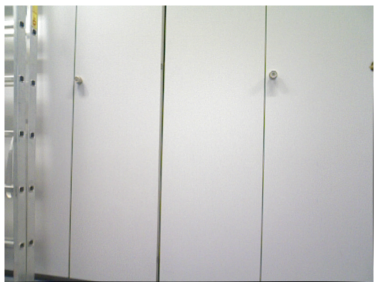
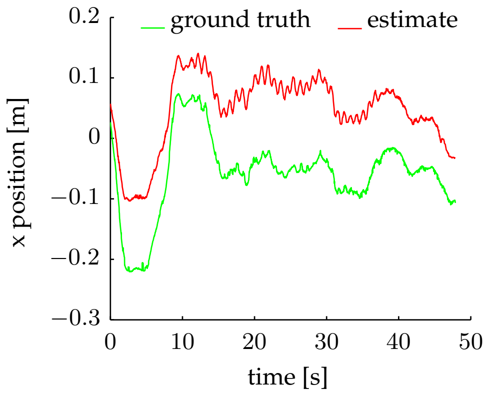
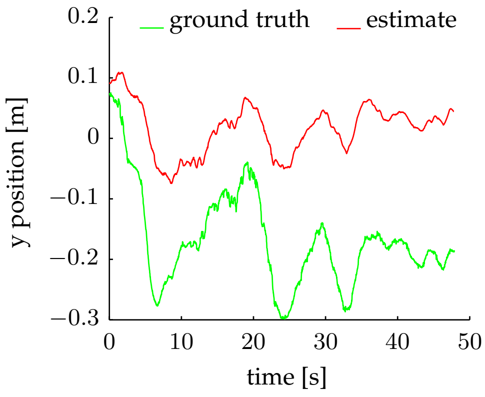

(c) mostly textureless and planar scene

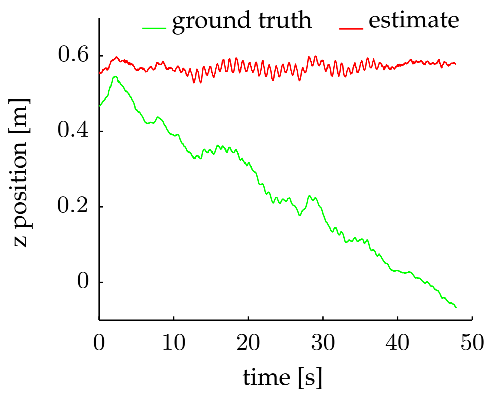
Figure 5.2.: Different scenes used for flight experiments. The images show the area, which is seen by the RGB-D camera and used for pose estimation. The scene content varies from texture and structure rich (a) to texture less and planar (c).



(a) x position

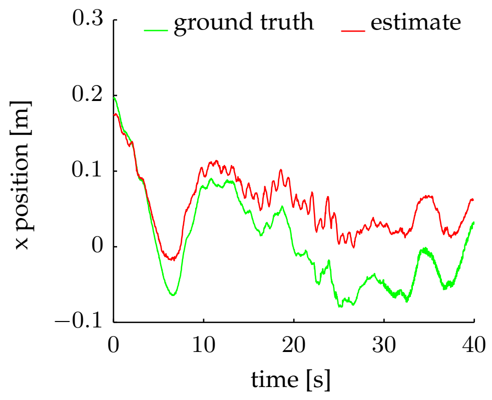


(b) y position

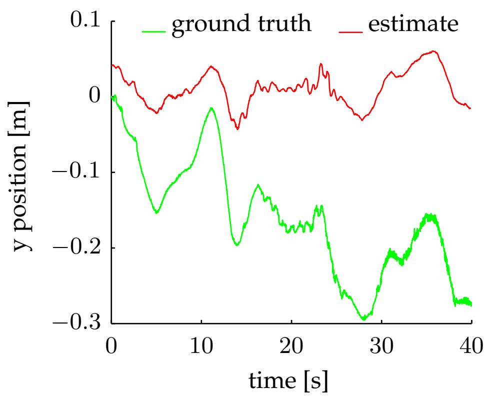


(c) z position

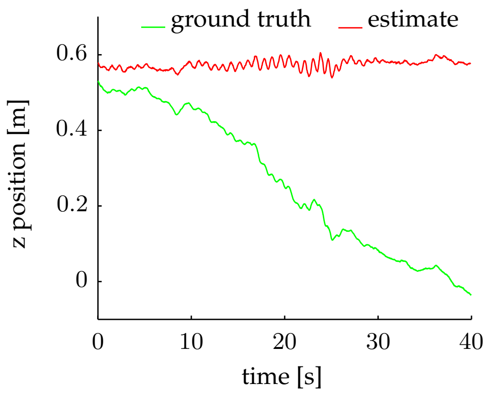
Figure 5.3.: Estimated vs. ground truth trajectory for a flight in the cluttered scene (see figure 5.2a) over 48 s. While the x and y component exhibit only little drift, the drift in z direction is severe.



(a) x position



(b) y position



(c) z position

Figure 5.4.: Estimated vs. ground truth trajectory for a flight in the normal scene (see figure 5.2b) over 40 s. While the x and y component exhibit only little drift, the drift in z direction is severe.

several trials are performed. Figure 5.3 and figure 5.4 show the trajectory of a representative flight on the two different scenes respectively. Every trajectory is split into separate plots for the x, y and z component with a graph for the ground truth position (green) and the estimated position (red). While there is a constant offset between ground truth and estimate for the x and y component, there is only little drift over time. In contrast, the z component exhibits severe drift in both experiments. For the texture rich scene the RMSE of the translational drift is 0.0298 m/s over a duration of 48 s. The second scene setup leads similarly to a RMSE translational drift of 0.0289 m/s. The length of this dataset is 40 s. These experiments validate the results of the evaluation on the synthetic and RGB-D benchmark datasets (cf. section 5.1). Furthermore, they show that the visual odometry approach is applicable on the quadrocopter and can be integrated into the control loop for position stabilization.

5.3. Parameter Optimization

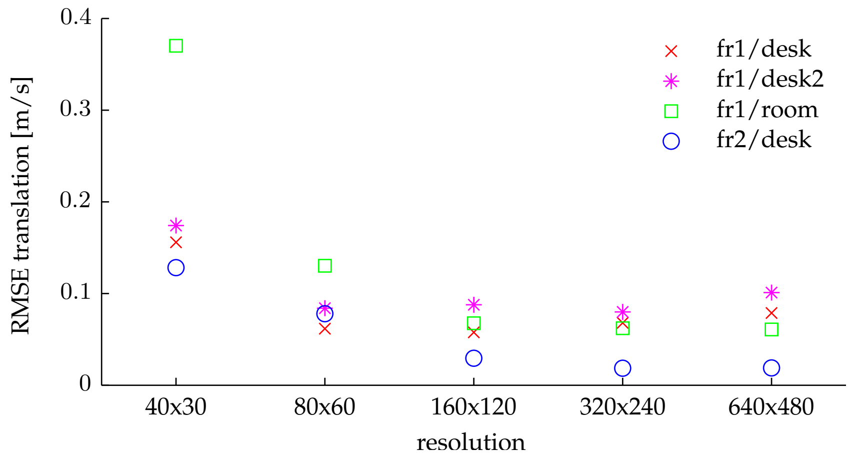
Several parameters control the optimization presented in algorithm 1. This section studies the influence of these parameters on the performance. Using different configurations the algorithm is executed on multiple datasets of the RGB-D benchmark [51]. Every run results in an estimated camera trajectory. Different tools provided by the RGB-D benchmark calculate metrics for the estimated trajectory in comparison to the ground truth trajectory. These metrics include the root mean square error (RMSE), mean, standard deviation, median, minimum, and maximum of the translational and rotational drift. The drift can be absolute or relative. The absolute drift states how far the final camera pose is from the final pose in the ground truth. The relative drift represents the deviation from the ground truth in a certain time frame. The following comparisons apply the RMSE of the relative translational and rotational drift per second.

The drift per second is a more reliable metric than the drift per frame used by other researchers [47, 50], because the noise in the ground truth of the RGB-D benchmark datasets has a similar order of magnitude [51]. For example, experiments, which simulate a static camera, i.e. not moving during the whole sequence, have a lower drift per frame than algorithms estimating the trajectory. Another difference to related publications is the use of the RMSE instead of the median. The RMSE is more sensitive to gross errors in the estimated trajectory than the median. Therefore, low RMSE values ensure good estimates over the whole trajectory.

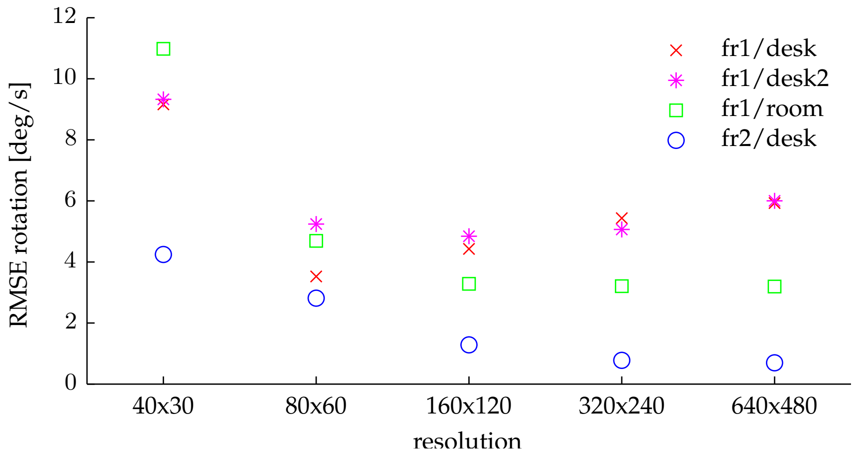
The main parameters are the coarsest level l_0, the finest level l_{max} and the precision ϵ. The maximum iterations parameter k_{max} is merely a safe guard to ensure the optimization finishes after a maximum time. The coarsest level l_0 defines the image resolution for starting the optimization and the finest level l_{max} specifies the resolution at which the algorithm terminates. In case l_0 and l_{max} are equal, the optimization runs on one resolution level only. A 2×2 block averaging filter extracts a coarser resolution from the next finer one. The highest resolution is 640×480 pixels and 320×240, 160×120, 80×60 and 40×30 pixels are the derived resolutions.

5.3.1. Resolution vs. Drift

The first experiment evaluates the dependence of the drift on the resolution used for optimization. The precision ϵ is fixed to 10^{-30} and the maximum number of iterations to 10000. Therefore, the only stop condition for the optimization is the increase in the error (cf. algorithm 1). Figure 5.5a shows the translational drift as a function of the resolution used for four datasets. Respectively, figure 5.5b depicts the dependency of the rotational drift. In both plots the error decreases with increasing resolution for the *fr1/room* and the *fr2/desk* sequence, although there is only a minor improvement from 320×240 to 640×480. In contrast, the drift has its minimum at 80×60 and 160×120 for the *fr1/desk* and *fr1/desk2* sequences. For higher resolutions the drift increases again. One reason is the higher average velocity in the *fr1/desk* (0.413 m/s) and the *fr1/desk2* (0.426 m/s) sequences in comparison to the *fr1/room* (0.334 m/s) and fr2/desk (0.193 m/s) sequences. The higher speed results in larger displacement between two images than on slower sequences.



(a) RMSE of translational drift



(b) RMSE of rotational drift

Figure 5.5.: Drift per second when optimizing on different resolutions. The drift decreases with increasing resolution on slow sequences, e.g. *fr2/desk*. On the contrary, the drift increases for higher resolutions on fast sequences since the displacement between consecutive images is larger.

5.3.2. Coarse-to-Fine vs. Drift

The second experiment studies the effect of optimizing over multiple resolution levels. The optimization starts at a low resolution and uses the estimate as initialization for the optimization on higher resolutions. This exploits the previous result that lower resolutions give good estimates for larger image displacements while higher resolutions provide estimates with less drift. In the experiment the optimization always starts at a resolution of 80×60 pixels and the number of higher resolutions varies. The precision and maximum iterations parameters are, as before, fixed to values not affecting the termination of the optimization.

Figure 5.6a displays the translational drift depending on the number of levels in the image pyramid, figure 5.6b depicts the rotational drift, respectively. The results for one pyramid level are equivalent to those for the 80×60 resolution in figure 5.5. In contrast to the previous experiment the drift reduces with the number of pyramid levels as higher resolutions are included in the optimization. The improvement from three to four levels, i.e. with and without 640×480 images, remains small.

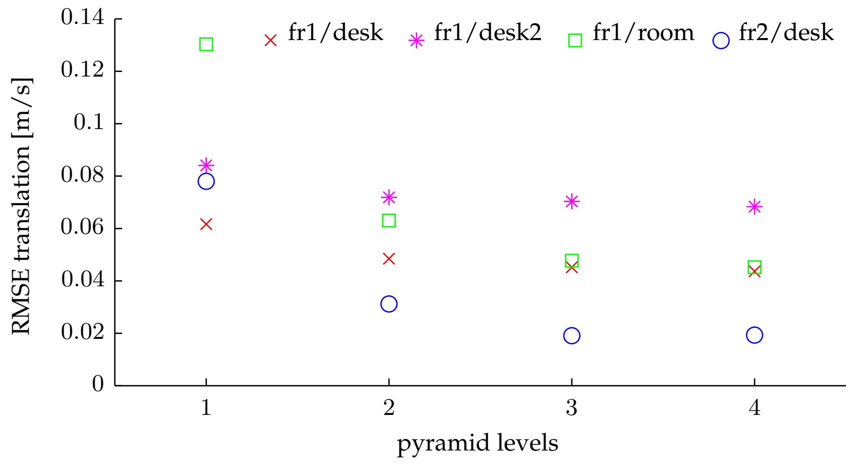
5.3.3. Precision vs. Drift

The next experiment evaluates the influence of the precision parameter ϵ. The parameter is varied between $5 \cdot 10^{-1}$ and 10^{-10}. This is done at four different resolutions ranging from 80×60 to 640×480. Figure 5.7 displays the translational and rotational drift for the different values of ϵ and different resolutions on the *fr1/desk* sequence. The supplementary figure 5.8 shows the average number of iterations with respect to the precision. In the range from $5 \cdot 10^{-1}$ to $5 \cdot 10^{-3}$ the drift is constant around $0.4\,\mathrm{m/s}$ due to stopping after the first iteration (see figure 5.8). Therefore, the error decrease attained by the first increment has to be between $5 \cdot 10^{-3}$ and 10^{-3}. The drift decreases with higher precisions until it plateaus for precisions beyond 10^{-6}. In contrast, the number of iterations still increases up to a precision of 10^{-9}. Concluding, error decrements below 10^{-7} have only little influence on the drift, but increase the computational load.

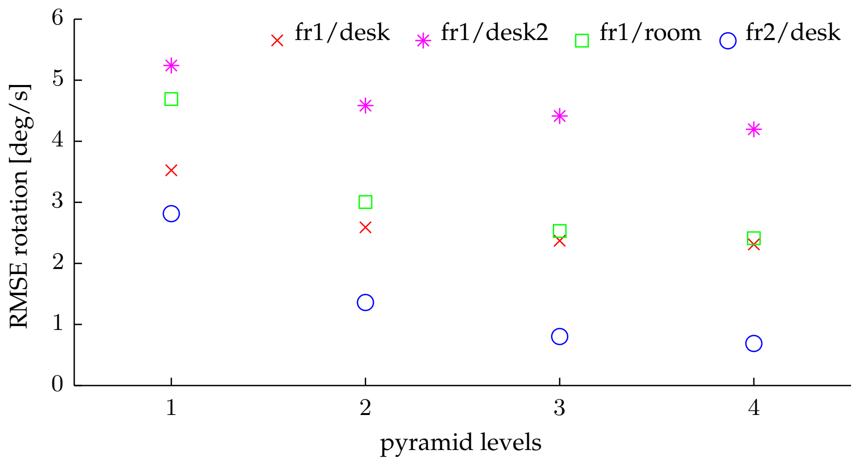
For lower resolutions fewer iterations are performed at a given precision. There are two possible reasons for this behavior. Either the error decrement is below the precision after fewer iterations, or the error increases stopping the optimization. Figure 5.9 shows the percentage of aborts due to error decrements below the precision for different resolutions. The complementary percentage is the amount of aborts due to error increase. It is dominant for lower resolutions at bigger values of ϵ. Even at the highest resolution the error increase is the only abort reason for $\epsilon \leq 10^{-10}$. Therefore, $\epsilon = 10^{-10}$ is the smallest sensible value.

5.3.4. Coarse-to-Fine and Precision vs. Drift

As the previous experiment shows, optimization over several resolution levels is beneficial to increase the accuracy of the approach. The following experiment evaluates the influence of the precision parameter on the number of iterations performed at each resolution and the resulting drift. The experiment uses the *fr1/desk* dataset. Figure 5.10 shows the value of ϵ and the respective average number of iterations when optimizing over four resolu-

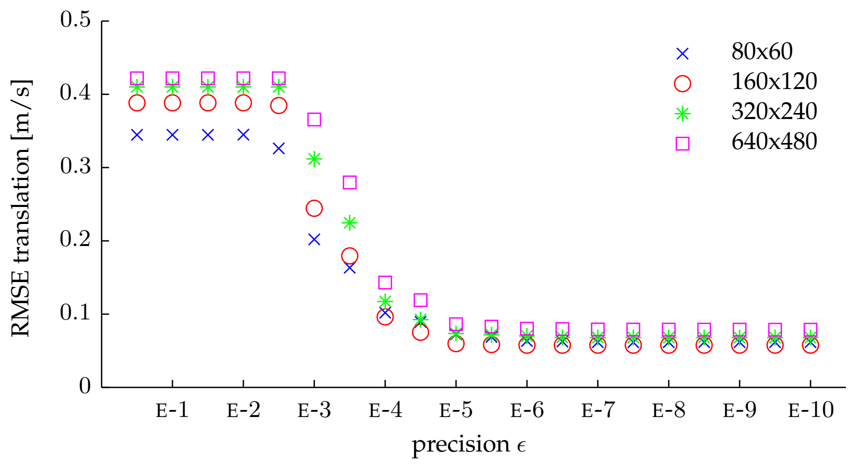


(a) RMSE of translational drift

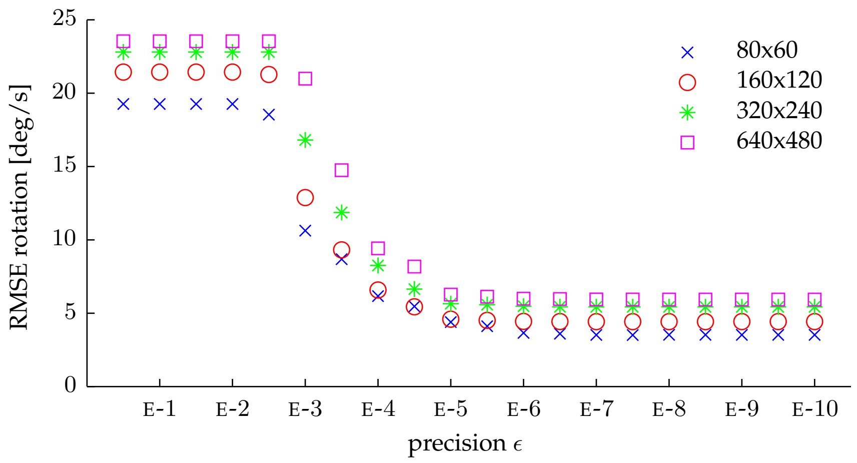


(b) RMSE of rotational drift

Figure 5.6.: Drift per second depending on the number of levels when optimizing over an image pyramid with different resolutions. The lowest level has a resolution of 80×60 pixels. A value of *pyramid levels* equal to one means optimization only on the lowest resolution. For the value four the optimization uses all resolutions up to 640×480 pixels. As more pyramid levels are used the drift decreases even for fast sequences.



(a) RMSE of translational drift



(b) RMSE of rotational drift

Figure 5.7.: Drift per second on the *fr1/desk* sequence for different values of the precision parameter ϵ and optimization using different resolutions. As the values get smaller the drift decreases. For $\epsilon \leq 10^{-6}$ there is only minor improvement.

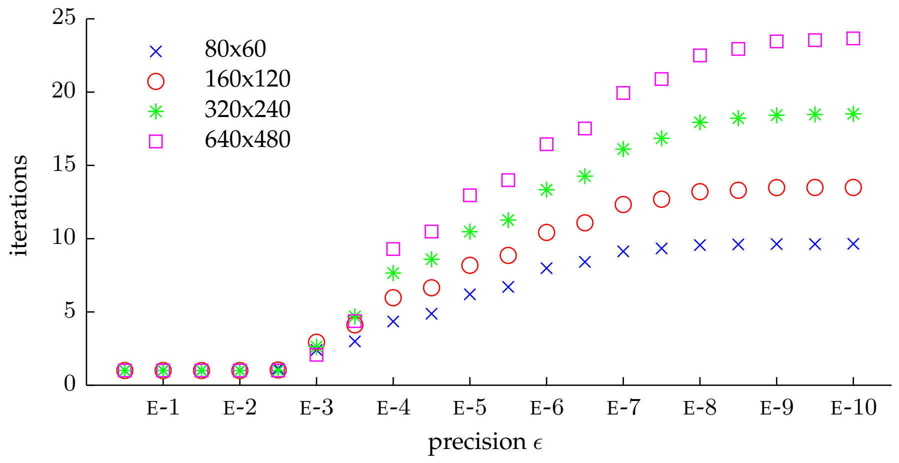


Figure 5.8.: The number of iterations required by the optimization for varying values of the precision ϵ and different resolutions. With lower values of ϵ the number of iterations increases. On higher resolutions more iterations are required until the optimization is stopped.

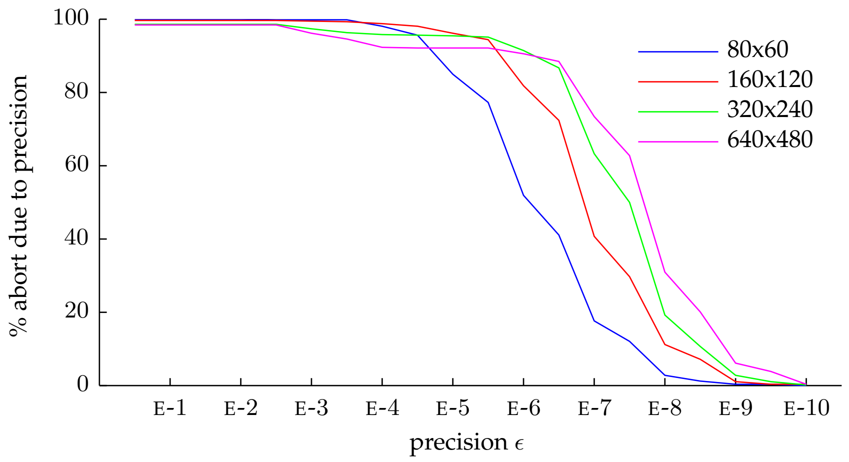


Figure 5.9.: The percentage of aborts due to the error decrement being below the given precision ϵ. The complementary percentage describes aborts because of increasing error. On lower resolutions the maximum precision before the error increases is less than on higher resolutions.

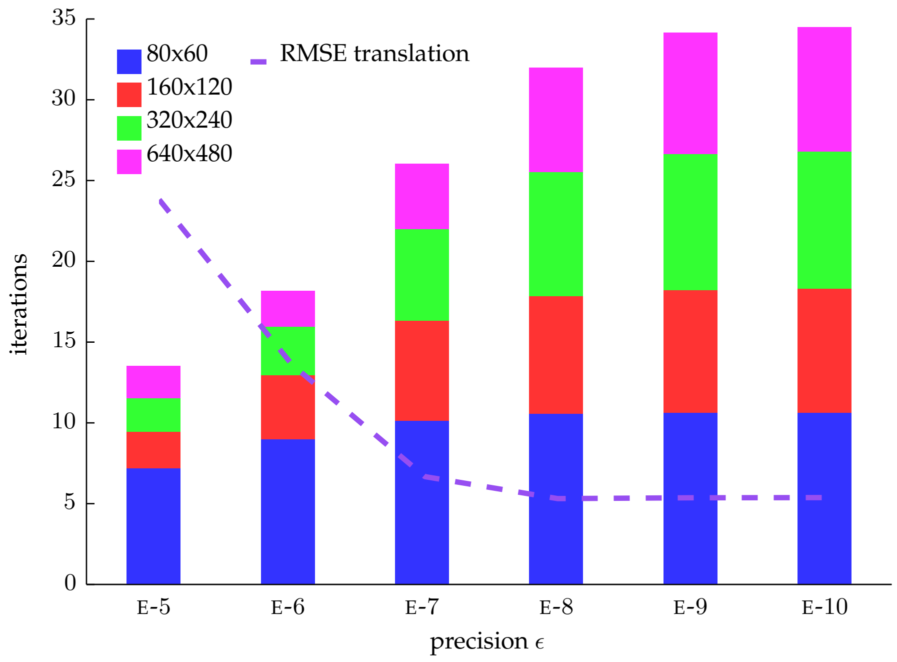


Figure 5.10.: The number of iterations required on the different resolutions when optimizing on an image pyramid to achieve a given precision ϵ. The qualitative trend of the RMSE of the translational drift is superimposed.

tion levels. The qualitative trend of the RMSE of the translational drift is superimposed. Setting $\epsilon = 10^{-5}$ requires only two iterations at higher resolutions. This indicates that the error already decreased to this order of magnitude on the lowest resolution, because the precision stop criterion requires at least two iterations (cf. algorithm 1). The lowest drift is obtained at a precision of 10^{-8} and stagnates for lower values of ϵ. The experiment proves that optimization over several resolutions not only improves the accuracy, but also reduces the number of iterations required at higher resolutions. This is advantageous because the number of pixels increases at every higher resolution by factor four and so does the number of computations.

5.3.5. T-distribution Degrees of Freedom vs. Drift

The degrees of freedom parameter ν of the t-distribution is determined by a parameter search on several datasets. The experiment varies the parameter in the interval $[1, 14]$ and computes the RMSE of translational and rotational drift on several datasets from the RGB-D benchmark. Figure 5.11 depicts the results. The x axis shows the value of ν and the y axis the normalized RMSE. The normalization divides every RMSE value by the mean of all RMSE values on the same data set but different ν values. Therefore, the shape of the graph is not altered, but the graphs of multiple datasets are easier to compare. Additionally, all normalized RMSE values corresponding to the same ν are averaged. The dashed lines depict the normalized and averaged RMSE drift when no weighting is used. The

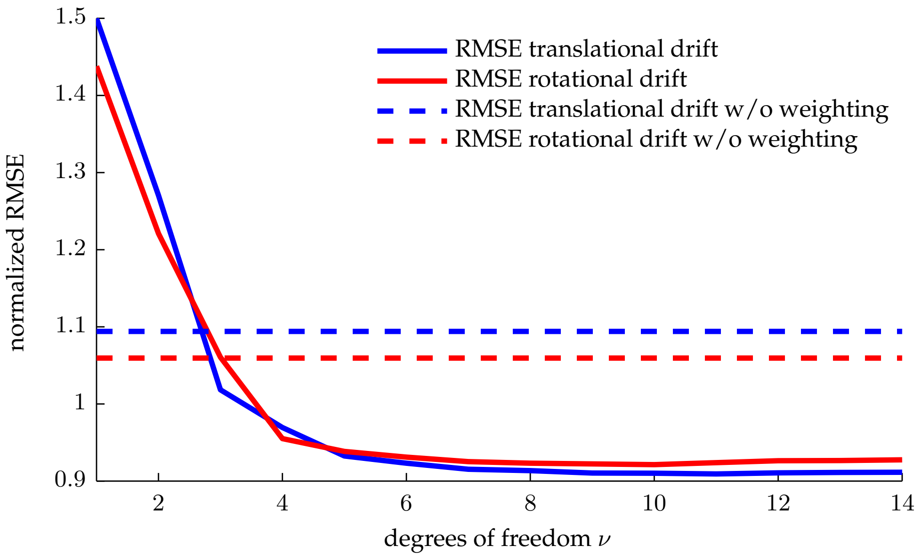


Figure 5.11.: The averaged, normalized RMSE drift values of different sequences for different choices of the degrees of freedom parameter ν. The dashed lines indicate the drift without weighting. The value $\nu = 5$ is a good trade-off between robustness and small drift.

plot shows that with $\nu \geq 3$ the weighted version performs better than the one without weighting. The improvement for values of ν larger than 5 is small. Values larger than 10 begin to decrease performance again. For larger values of ν the RMSE graph approaches the line, which represents the unweighted case. Outliers get more strongly downweighted for smaller values of ν (c.f. equation 4.31) [33, 15]. Therefore, the degrees of freedom parameter ν is fixed to 5 as this value ensures low RMSE drift and good robustness.

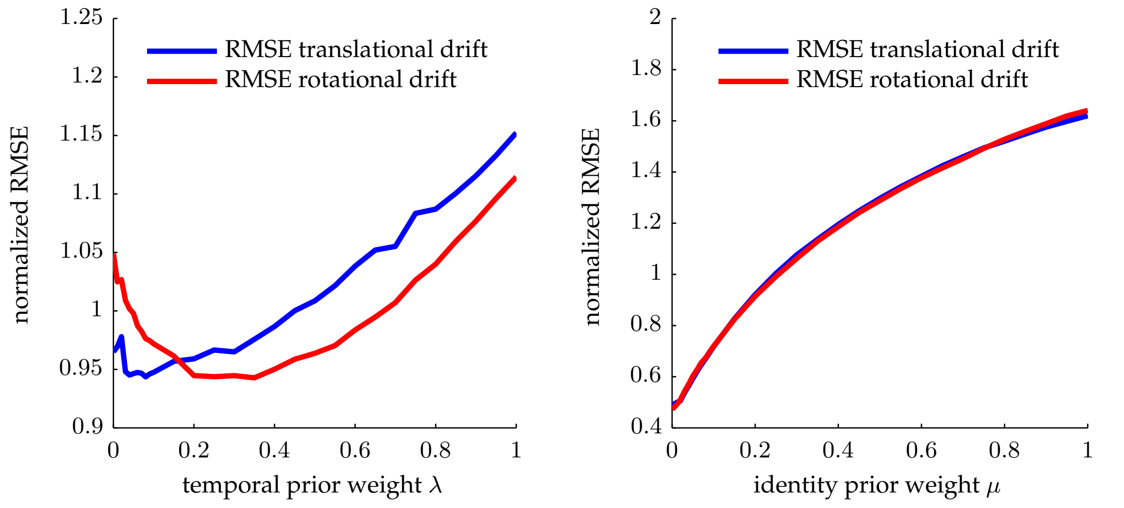
5.3.6. Priors vs. Drift

The parameters of the priors detailed in section 4.3.2 are determined in similar experiments as the degrees of freedom parameter ν. For different values of the weighting parameters the RMSE of the translational and rotational drift per second is calculated on a number of RGB-D benchmark datasets. The parameter values vary in the interval $[0, 1]$. The RMSE values are normalized and averaged for comparison.

Figure 5.12a shows the results for the temporal prior with parameter λ. The x axis depicts the values of λ and the y axis the corresponding, normalized, and averaged RMSE value. For the translational drift the optimal λ is between 0.03 and 0.08. The rotational drift is minimal for λ in the range 0.2 and 0.35. Both graphs intersect for $\lambda = 0.16$.

Figure 5.12b displays the same experiment for the identity prior with parameter μ. As the drift monotonically grows with increasing μ it appears that the identity prior has no positive effect on the RMSE. Therefore, it can be discarded.

Section 4.3.2 discusses the use of prior information only as initialization. The following experiment evaluates the dependency of the drift on the initialization. It tests two



(a) drift for different temporal prior weights λ (b) drift for different identity prior weights μ

Figure 5.12.: Rotational and translational drift for different weight factors of the temporal prior (a) and identity prior (b). Incorporating the temporal prior with small values of λ improves performance. The identity prior always increases the drift.

initializations. The first initialization is the identity and the second is the estimated transformation between the previous RGB-D image pair. Figure 5.13 shows the results. The x axis depicts the different datasets and the y axis the RMSE of the translational drift. The performance of the initialization with the previous estimate is worse on all datasets except *fr3/sitting_halfsphere* than the initialization with the identity. Therefore, the previous estimate seems to be a bad initial guess for the optimization.

The last two experiments provide two insights. First the use of the temporal prior through $p(\boldsymbol{\xi})$ is beneficial to reduce drift for small values of the weighting parameter λ. In contrast, initialization with the last estimate degrades the performance. The second insight is, that the contrary argument to the temporal prior holds for the identity prior. It is good for initialization, but increases drift when used as $p(\boldsymbol{\xi})$.

5.3.7. Runtime

Table 5.5 summarizes the computational time required for one iteration at different resolutions. The second column shows the timings with disabled weighting and the third column with enabled weighting. The execution time is measured on a PC with an Intel i5 670 CPU (3.46 GHz) and 4 GB RAM. For each higher resolution level the time per iteration increases by factor four approximately. Enabling robust weighting barely affects the runtime.

Table 5.6 shows the approximate runtime for the alignment of one image pair computed from the timings in table 5.5 and the average number of iterations in figure 5.10. The second column displays the runtimes excluding the highest resolution and the third column the runtimes including it. As new RGB-D images arrive at 30 Hz (see section 2.2.2) the maximum time for processing an image pair is 33.3 ms. As the table shows including

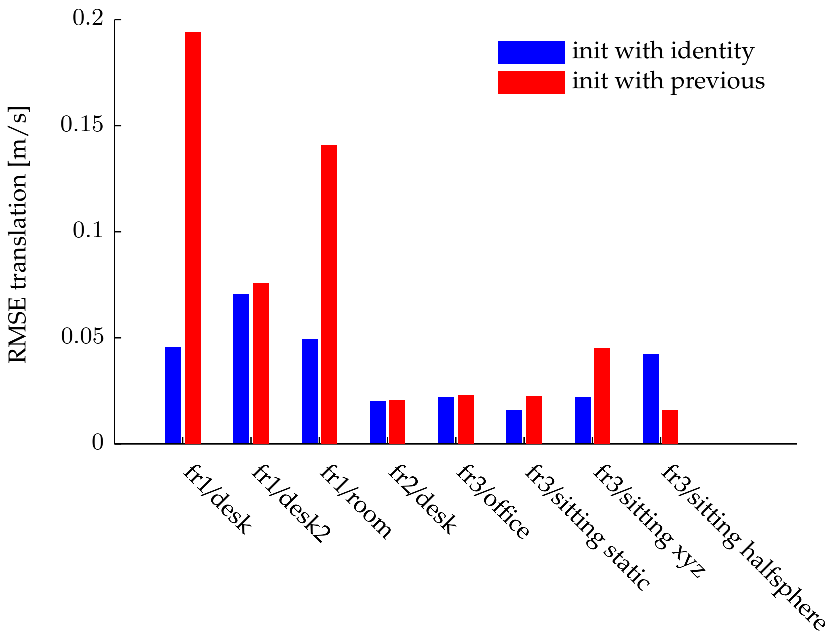


Figure 5.13.: Comparison of the translational drift depending on the initialization for different datasets. Initialization with the previous estimate increases drift compared to initialization with the identity.

Table 5.5.: Average runtimes per iteration with weighting enabled and disabled for different resolutions. Between a resolution and the next higher one the runtime increases approximately by factor 4. Enabling weighting has only a minor impact on the runtime.

resolution	∅ runtime/iteration without weighting [ms]	∅ runtime/iteration with weighting [ms]
80×60	0.35	0.37
160×120	1.03	1.08
320×240	4.26	4.42
640×480	17.24	17.78

Table 5.6.: Approximate runtimes required for the alignment of two RGB-D images with different values of ϵ and l_{max}. The approximation is based on the timings per iteration in table 5.5 and the average number of iterations in figure 5.10. Optimization including the highest resolution results in runtimes not usable in realtime scenarios (maximum runtime 33.3 ms). When the highest resolution is chosen as 320×240 and ϵ between 10^{-6} and 10^{-7} the highest accuracy is attained while staying in the realtime limit.

precision ϵ	approx. runtime highest resolution 320×240 [ms]	approx. runtime highest resolution 640×480 [ms]
10^{-5}	14.15	49.74
10^{-6}	20.96	60.19
10^{-7}	35.49	107.48
10^{-8}	45.63	160.55
10^{-9}	49.44	183.10
10^{-10}	49.68	186.61

the highest resolution is prohibitively expensive in terms of runtime. The best accuracy without violating the realtime constraint is obtained for ϵ between 10^{-6} and 10^{-7} and a maximum resolution of 320×240 pixels. As the previous experiments highlight the use of the highest resolution yields only minimal improvement in terms of drift.

5.3.8. Summary

The experiments in this section show that optimization in a coarse to fine manner, starting at a low resolution and using the estimate as initialization for optimization on higher resolution, is superior in terms of accuracy and runtime to optimization on a single resolution. Therefore, the coarsest level l_0 is set to 80×60 and the finest level l_{max} to 320×240. Furthermore, the precision as threshold on the error decrease between two iterations allows to trade off performance against accuracy. The value $\epsilon = 5 \cdot 10^{-7}$ results in high accuracy while the runtime remains within the realtime limit of 33 ms. The maximum number of iterations is limited to $k_{\text{max}} = 100$. The degrees of freedom parameter ν of the t-distribution is fixed to 5 which provides a good trade-off between robustness and drift reduction. For the different priors the experiments show that the initialization with the identity is favorable. In contrast, the temporal prior is beneficial when used as $p(\boldsymbol{\xi})$ with a weighting parameter λ in the range from 0.05 to 0.3.

6. Conclusion

This thesis has developed a robust, dense visual odometry approach which utilizes the data provided by recent RGB-D cameras. The robustness has been achieved by giving a Bayesian derivation of the underlying least squares problem and modeling the distributions based on empirical data. The t-distribution is a good model. Additionally, this derivation allows to incorporate prior knowledge about the motion, e.g. from other sensors, naturally. Due to the dense nature of the approach using all image information very high accuracy over long trajectories can be achieved. This performance has been validated trough extensive experiments on real and synthetic data. The drift of the estimated trajectory is in the range of 2-4 cm/s. At the same time the implementation is capable of running at camera frame rate on a single core of a recent commodity CPU. Furthermore, experiments on the quadrocopter show that the visual odometry approach can be used to stabilize the position over several seconds without human intervention. While long time operation can be achieved by a human operator correcting for the accumulating drift, fully autonomous operation remains an open challenge.

There are several possible directions for future research:

- The major violations of the photo-consistency assumption stem from automatic adaption of the camera exposure and gain. These cause a shift in the brightness of the whole image, which results in very large residuals all over the image. Observations show that this leads to large jumps in the estimated trajectory. By modeling these effects and estimating the parameters simultaneously to the motion the brightness changes could be compensated [28].

- The integration of the depth error into the minimization as already proposed by Tykkälä et al. [54] could further improve accuracy, especially in regions with few texture. Instead of using an entirely depth based error metric a point-to-plane error as in ICP could be used [58]. An open problem with this simultaneous optimization is to find a strategy how to downweight outliers in both error terms, which are only identified either in the intensity image or the depth image.

- An alternative to aligning consecutive frames is to match against a sufficiently close keyframe. This could further reduce drift and jitter during slow motions [24]. The keyframes can be replaced with keyframe models into which successfully matched RGB-D frames are integrated to reduce noise. From this point the next step could be the extension to a full SLAM system, e.g. including pose graph optimization.

- The optimization only requires the depth information of the reference image. Therefore, tracking a monocular camera against an existing textured 3D model of the scene should be possible.

- For the minimization alternate optimization schemes like Efficient Second Order Minimization (ESM) [39] could be evaluated. It might provide faster convergence, but at the same time increase computational requirements.
- The robustness against dynamic objects occupying a large amount of the field of view remains challenging. Possibly, these objects could be identified more easily in depth images due to their large changes in depth.
- The use of data from the IMU as initialization or prior for the optimization is discussed in section 4.3.2, but it is not implemented due to inter-sensor pose calibration and time synchronization issues. Especially the rotation measurements could be useful.
- An evaluation of the approach using passive stereo cameras would be interesting, because it would relax the limitation to indoor environments.

In conclusion, this thesis shows that it is possible to implement a robust, dense visual odometry approach on currently available quadrocopter hardware, and use it for control and stabilization. Further, the consequent Bayesian treatment of the optimization problem provides a sound and natural framework for incorporating robustness.

A. RGB-D Benchmark Detailed Results

Table A.1.: Detailed results for the *Testing and Debugging* datasets.

Method	fr1 xyz	fr1 rpy	fr2 xyz	fr2 rpy	∅Drift [m/s]	Improvement
reference	0.0365	0.1049	0.0127	0.0175	0.0429	0.00 %
no weights	0.0438	**0.0331**	0.0131	0.0147	0.0262	38.98 %
t-dist. weights	0.0331	0.0336	0.0133	0.0177	0.0244	43.10 %
t-dist. weights+tmp	**0.0308**	0.0372	0.0125	0.0165	0.0243	43.43 %
huber weights	0.0340	0.0333	**0.0113**	**0.0145**	0.0233	45.74 %
tukey weights	0.1315	0.1003	0.0417	0.0445	0.0795	-85.26 %

Table A.2.: Detailed results for the *Handheld SLAM* datasets.

Method	fr1 desk	fr1 desk2	fr1 room	fr2 desk	fr3 office	∅Drift [m/s]	Improvement
reference	0.5370	0.3416	0.9420	0.0205	**0.0163**	0.3715	0.00 %
no weights	0.0551	0.1003	0.0709	0.0231	0.0182	0.0535	85.59 %
t-dist. weights	**0.0458**	**0.0708**	0.0496	0.0203	0.0222	0.0417	88.77 %
t-dist. weights+tmp	0.0459	0.0713	0.0520	**0.0186**	0.0201	0.0416	88.81 %
huber weights	0.0476	0.0805	**0.0491**	**0.0186**	0.0313	0.0454	87.77 %
tukey weights	0.1740	0.2072	0.2409	0.1080	0.1703	0.1801	51.52 %

Table A.3.: Detailed results for the *Robot SLAM* datasets.

Method	fr2 p. 360	fr2 p. slam	fr2 p. slam2	fr2 p. slam3	∅Drift [m/s]	Improvement
reference	27.2764	1.5376	4.5239	0.9866	8.5811	0.00 %
no weights	0.2196	0.3086	0.2021	0.1518	0.2205	97.43 %
t-dist. weights	0.2820	0.2906	0.2943	0.1532	0.2550	97.03 %
t-dist. weights+tmp	**0.0892**	**0.1024**	**0.0897**	**0.0476**	0.0822	99.04 %
huber weights	0.2728	0.2787	0.2980	0.1227	0.2431	97.17 %
tukey weights	0.3834	0.3529	0.3753	0.2689	0.3451	95.98 %

Table A.4.: Detailed results for the *sitting* datasets in the *Dynamic Objects* category.

Method	desk w/ person	static	xyz	rpy	half-sphere	static (v)	xyz (v)	rpy (v)	half-sphere (v)	∅Drift [m/s]	Improve-ment
reference	0.0708	0.0213	0.0284	0.1411	0.1997	0.0307	0.0309	0.1104	0.0779	0.0790	0.00 %
no weights	0.0567	0.0199	0.0259	0.0418	0.0444	0.0231	0.0296	0.0641	0.0315	0.0374	52.61 %
t-dist. weights	0.0360	0.0161	0.0221	0.0786	0.0424	0.0184	0.0275	0.1415	0.0290	0.0457	42.10 %
t-dist. weights+tmp	**0.0347**	0.0165	0.0215	**0.0407**	**0.0421**	0.0173	0.0262	**0.0570**	**0.0261**	0.0314	60.32 %
huber weights	0.0397	**0.0160**	**0.0216**	0.0580	0.0448	**0.0174**	**0.0256**	0.1419	0.0300	0.0439	44.46 %
tukey weights	0.1073	0.0283	0.0741	0.1309	0.1424	0.0309	0.1175	0.1740	0.1323	0.1042	-31.85 %

Table A.5.: Detailed results for the *walking* datasets in the *Dynamic Objects* category.

Method	static	xyz	rpy	half-sphere	static (v)	xyz (v)	rpy (v)	half-sphere (v)	∅Drift [m/s]	Improve-ment
reference	0.4321	0.5622	0.6380	0.5156	0.4783	0.6263	0.4870	0.4085	0.5185	0.00 %
no weights	0.3438	0.4616	0.4851	0.4294	0.4123	0.5363	0.4083	0.3639	0.4301	17.05 %
t-dist. weights	0.3433	0.5349	0.5446	0.4727	0.4393	0.5866	0.4511	0.4050	0.4722	8.93 %
t-dist. weights+tmp	0.3498	0.5381	0.5396	0.4660	0.4730	0.5862	0.4456	0.4084	0.4758	8.23 %
huber weights	0.3414	0.5131	0.5245	0.4611	0.4335	0.5734	0.4416	0.3995	0.4610	11.09 %
tukey weights	0.0635	0.3212	0.4119	0.3146	0.0823	0.3347	0.2963	0.2928	0.2647	48.96 %

Bibliography

[1] Natural Human-Robot Team NIFTi Successfully Aids in Recovery from Earthquake in Emilia-Romagna, Italy. `http://www.nifti.eu/news/official-press-release-of-dfki-english`, 2012. Accessed November 06, 2012.

[2] Photo album of Mirandola mission on Facebook. `http://www.nifti.eu/news/photo-album-of-mirandola-mission-on-facebook`, 2012. Accessed November 10, 2012.

[3] S. Baker and I. Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. *Intl. Journal of Computer Vision*, 56(3):221–255, March 2004.

[4] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, June 2008.

[5] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

[6] M.J. Black, D.J. Fleet, and Y. Yacoob. Robustly Estimating Changes in Image Appearance. *Computer Vision and Image Understanding*, 78:8–31, 2000.

[7] G. Blais and M.D. Levine. Registering multiview range data to create 3D computer objects. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):820–824, August 1995.

[8] A.I. Comport, E. Malis, and P. Rives. Real-time Quadrifocal Visual Odometry. *Intl. Journal of Robotics Research*, 29(2–3):245–266, February 2010.

[9] A.I. Comport, E. Marchand, and F. Chaumette. Statistically robust 2D visual servoing. *IEEE Transactions on Robotics*, 22(2):415–421, April 2006.

[10] J. Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors, 2006.

[11] J. Engel, J. Sturm, and D. Cremers. Camera-Based Navigation of a Low-Cost Quadrocopter. In *Proc. of the Intl. Conference on Intelligent Robot Systems (IROS)*, October 2012.

[12] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard. Real-time 3D visual SLAM with a hand-held camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, April 2011.

[13] F. Fraundorfer and D. Scaramuzza. Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications. *Robotics Automation Magazine, IEEE*, 19(2):78–90, June 2012.

[14] J. Gai and R.L. Stevenson. Studentized Dynamical System for Robust Object Tracking. *IEEE Transactions on Image Processing*, 20(1):186–199, January 2011.

[15] A. Gelman, J.B. Carlin, H.S. Stern, and D.B. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC, 2003.

[16] D. Gerogiannis, C. Nikou, and A. Likas. Robust Image Registration using Mixtures of t-distributions. In *Proc. of the Intl. Conference on Computer Vision (ICCV)*, pages 1–8. IEEE, 2007.

[17] Ascending Technologies GmbH. AscTec Pelican - Ascending Technologies. `http://www.asctec.de/uav-applications/research/products/asctec-pelican/`. Accessed November 04, 2012.

[18] E. Guizzo. Robotic Aerial Vehicle Captures Dramatic Footage of Fukushima Reactors. `http://spectrum.ieee.org/automaton/robotics/industrial-robots/robotic-aerial-vehicle-at-fukushima-reactors`, 2011. Accessed November 06, 2012.

[19] F.R. Hampel, E.M. Ronchetti, P.J. Rousseeuw, and W.A. Stahel. *Robust Statistics: The Approach Based on Influence Functions*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2011.

[20] J. Hampton. Flying drone gives inside scoop on cathedral damage. `http://www.3news.co.nz/Flying-drone-gives-inside-scoop-on-cathedral-damage/tabid/423/articleID/215242/Default.aspx`, 2011. Accessed November 6, 2012.

[21] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. of the Fourth Alvey Vision Conference*, pages 147–151, 1988.

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

[23] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments, 2010.

[24] A.S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera. In *Proc. of the Intl. Symposium of Robotics Research (ISRR)*, 2011.

[25] P.J. Huber. *Robust Statistics*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2003.

[26] ASUSTek Computer Inc. ASUS - Multimedia - ASUS Xtion PRO LIVE. `http://www.asus.com/Multimedia/Motion_Sensor/Xtion_PRO_LIVE/`. Accessed July 22, 2012.

[27] R. Jiang, R. Klette, and S. Wang. Modeling of Unbounded Long-Range Drift in Visual Odometry. In *Fourth Pacific-Rim Symposium on Image and Video Technology (PSIVT)*, pages 121–126, November 2010.

[28] H. Jin, P. Favaro, and S. Soatto. Real-time feature tracking and outlier rejection with changes in illumination. In *Proc. of the Eighth IEEE Intl. Conference on Computer Vision (ICCV)*, volume 1, pages 684–689, 2001.

[29] K. Khoshelham and S.O. Elberink. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. *Sensors*, 12(2):1437–1454, 2012.

[30] Y. Kim, A.M. Martínez, and A.C. Kak. Robust motion estimation under varying illumination. *Image and Vision Computing*, 23(4):365–375, April 2005.

[31] G. Klein and D. Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *Proc. of the Sixth IEEE and ACM Intl. Symposium on Mixed and Augmented Reality (ISMAR)*, November 2007.

[32] K. Konolige, M. Agrawal, and J. Solà. Large Scale Visual Odometry for Rough Terrain. In *Proc. of the Intl. Symposium on Research in Robotics (ISRR)*, November 2007.

[33] K.L. Lange, R.J.A. Little, and J.M.G. Taylor. Robust Statistical Modeling Using the t Distribution. *Journal of the American Statistical Association (JASA)*, 84(408), 1989.

[34] C. Liu and D.B. Rubin. ML estimation of the t distribution using EM and its extensions, ECM and ECME. *Statistica Sinica*, 5:19–39, 1995.

[35] D.G. Lowe. Object recognition from local scale-invariant features. In *Proc. of the Seventh IEEE Intl. Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157, 1999.

[36] J. Loxam and T. Drummond. Student-t Mixture Filter for Robust, Real-Time Visual Tracking. In *Proc. of the 10th European Conference on Computer Vision (ECCV)*, pages 372–385, 2008.

[37] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proc. of the 7th Intl. Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, pages 674–679, 1981.

[38] Y. Ma, S. Soatto, J. Kosecká, and S.S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Interdisciplinary applied mathematics: Imaging, vision, and graphics. Springer, 2003.

[39] E. Malis. Improving vision-based control using efficient second-order minimization techniques. In *Proc. of the IEEE Intl. Conference on Robotics and Automation (ICRA)*, volume 2, pages 1843–1848, May 2004.

[40] R.A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A.J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Proc. of the 10th IEEE Intl. Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011.

[41] R.A. Newcombe, S. Lovegrove, and A.J. Davison. DTAM: Dense tracking and mapping in real-time. In *Proc. of the Intl. Conference on Computer Vision (ICCV)*, pages 2320–2327, 2011.

[42] D. Nistér, O. Naroditsky, and J.R. Bergen. Visual Odometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–659, 2004.

[43] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision (ECCV)*, volume 1, pages 430–443, May 2006.

[44] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proc. of the Third Intl. Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[45] D. Scaramuzza and F. Fraundorfer. Visual Odometry: Part I: The First 30 Years and Fundamentals. *Robotics Automation Magazine, IEEE*, 18(4):80–92, December 2011.

[46] B. Siciliano and O. Khatib. *Springer Handbook of Robotics*. Gale virtual reference library. Springer, 2008.

[47] F. Steinbrücker, J. Sturm, and D. Cremers. Real-Time Visual Odometry from Dense RGB-D Images. In *Workshop on Live Dense Reconstruction with Moving Cameras at the Intl. Conference on Computer Vision (ICCV)*, 2011.

[48] C.V. Stewart. Robust Parameter Estimation in Computer Vision. *SIAM Reviews*, 41, 1999.

[49] H. Strasdat, J. M. M. Montiel, and A. Davison. Scale Drift-Aware Large Scale Monocular SLAM. In *Proc. of Robotics: Science and Systems*, June 2010.

[50] J. Stückler and S. Behnke. Model Learning and Real-Time Tracking Using Multi-Resolution Surfel Maps. In *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[51] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A Benchmark for the Evaluation of RGB-D SLAM Systems. In *Proc. of the Intl. Conference on Intelligent Robot Systems (IROS)*, 2012.

[52] R. Szeliski. *Computer Vision*. Springer London, 2011.

[53] P. H. S. Torr and D. W. Murray. The Development and Comparison of Robust Methods for Estimating the Fundamental Matrix. *Intl. Journal of Computer Vision*, 24(3):271–300, September 1997.

[54] T.M. Tykkälä, C. Audras, and A.I Comport. Direct Iterative Closest Point for Real-time Visual Odometry. In *The Second Intl. Workshop on Computer Vision in Vehicle Technology: From Earth to Mars in conjunction with the Intl. Conference on Computer Vision (ICCV)*, November 2011.

[55] S. Weiss, M. Achtelik, M. Chli, and R. Siegwart. Versatile Distributed Pose Estimation and Sensor Self-Calibration for an Autonomous MAV. In *Proc. of the IEEE Intl. Conference on Robotics and Automation (ICRA)*, 2012.

[56] S. Weiss, M. Achtelik, S. Lynen, M. Chli, and R. Siegwart. Real-time Onboard Visual-Inertial State Estimation and Self-Calibration of MAVs in Unknown Environments. In *Proc. of the IEEE Intl. Conference on Robotics and Automation (ICRA)*, 2012.

[57] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic Huber-L1 Optical Flow. In *Proc. of the British Machine Vision Conference (BMVC)*, September 2009.

[58] T. Whelan, H. Johannsson, M. Kaess, J.J. Leonard, and J.B. McDonald. Robust Tracking for Real-Time Dense RGB-D Mapping with Kintinuous. Technical Report MIT-CSAIL-TR-2012-031, Computer Science and Artificial Intelligence Laboratory, MIT, September 2012.

[59] Wikipedia. Quadrotor — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/w/index.php?title=Quadrotor&oldid=520172611`, 2012. Accessed November 06, 2012.

[60] Z. Zhang. Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting. *Image and Vision Computing*, 15(1):59–76, January 1997.

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