Efficient Visual SLAM for Autonomous Aerial Vehicles

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Dipl.-Inform. Sebastian A. Scherer
aus Waiblingen

Tübingen
2016

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	12. Dezember 2016
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Andreas Zell
2. Berichterstatter:	Prof. Dr. Andreas Schilling
3. Berichterstatter:	Prof. Dr. Horst-Michael Groß

To my parents, Eberhard and Elenore

Abstract

The general interest in autonomous or semi-autonomous micro aerial vehicles (MAVs) is strongly increasing. There are already several commercial applications for autonomous micro aerial vehicles and many more being investigated by both research institutes and multiple financially strong companies. Most commercially available applications, however, are rather limited in their autonomy: They rely either on a human operator or reliable reception of global positioning system (GPS) signals for navigation.

Truly autonomous micro aerial vehicles that can also fly in GPS-denied environments such as indoors, in forests, or in urban scenarios, where the GPS signal may be blocked by tall buildings, clearly require more on-board sensing and computation potential. In this dissertation, we explore autonomous micro aerial vehicles that rely on a so-called RGBD camera as their main sensor for simultaneous localization and mapping (SLAM). Several aspects of efficient visual SLAM with RGBD cameras aimed at micro aerial vehicles are studied in detail within this dissertation:

We first propose a novel principle of integrating depth measurements within visual SLAM systems by combining both 2D image position and depth measurements. We modify a widely-used visual odometry system accordingly, such that it can serve as a robust and accurate odometry system for RGBD cameras.

Based on this principle we go on and implement a full RGBD SLAM system that can close loops and perform global pose graph optimization and runs in real-time on the computationally constrained onboard computer of our MAV.

We investigate the feasibility of explicitly detecting loops using depth images as opposed to intensity images with a state of the art hierarchical bag of words (BoW) approach using depth image features.

Since an MAV flying indoors can often see a clearly distinguishable ground plane, we develop a novel efficient and accurate ground plane detection method and show how to use this to suppress drift in height and attitude.

Finally, we create a full SLAM system combining the earlier ideas that enables our MAV to fly autonomously in previously unknown environments while creating a map of its surroundings.

Kurzfassung

Das allgemeine Interesse an autonomen oder halb-autonomen kleinen unbemannten Flugzeugen, auch als Micro Aerial Vehicles (MAV) bekannt, wächst stetig. Es gibt bereits mehrere kommerzielle Anwendungen solcher autonomer Flugobjekte und viele weitere sind aktuell in der Entwicklung: Nicht nur in Forschungsinstituten, sondern auch bei einigen finanzstarken Firmen. Die meisten bereits existierenden Lösungen sind in ihrer Autonomie allerdings noch immer stark eingeschränkt: Sie erfordern entweder einen menschlichen Piloten am Boden oder den zuverlässigen Empfang eines GPS-Signals zur sicheren Navigation.

Wirklich autonome unbemannte Flugobjekte, die auch in Umgebungen ohne GPS-Empfang fliegen können, also im Inneren von Häusern, in Wäldern, oder auch in Stadtgebieten mit hohen Gebäuden, die das GPS-Signal abschirmen können, benötigen mehr Sensorik und damit auch mehr Rechenleistung an Bord. Diese Dissertation beschäftigt sich mit unbemannten Flugobjekten, die eine sogenannte RGBD-Kamera als Hauptsensor zur visuellen Selbstlokalisierung bei gleichzeitiger Kartierung (Simultaneous Localization and Mapping, SLAM) besitzen. Weiterhin untersuchen wir diverse Aspekte effizienter visueller SLAM-Algorithmen für RGBD Kameras im Hinblick auf den Einsatz auf kleinen unbemannten Flugobjekte:

Zunächst stellen wir eine neuartige Methode vor, um Tiefenmessungen in visuellem SLAM zu berücksichtigen, indem 2D-Bildpositionen und Tiefenwerte fusioniert werden. Wir passen ein weit verbreitetes Verfahren für visuelle Odometrie entsprechend an, damit es als ein robustes und genaues Odometriesystem auch für RGBD-Kameras eingesetzt werden kann. Basierend auf dieser Idee implementieren wir ein vollständiges RGBD-SLAM-System, welches auch den globalen Posengraphen in Echtzeit auf dem Bordcomputer des MAVs optimieren kann. Wir untersuchen die Machbarkeit expliziter Schleifenerkennung unter Verwendung von Tiefen- statt Intensitätsbildern mit einem aktuellen hierarchischen Bag-of-Words-Ansatz mit Tiefenfeatures. Da bei Flügen im Innenbereich oft eine klar erkennbare Bodenebene zu sehen ist, präsentieren wir eine neuartige Methode, um diese schnell und exakt zu schätzen, um damit den Drift in Höhe und Orientierung zu verringern. Zuletzt entwickeln wir schließlich ein vollständiges SLAM-System, das die zuvor genannten individuellen Beiträge kombiniert und es dem MAV ermöglicht, in unbekannten Umgebungen autonom zu fliegen und zugleich eine Karte seiner Umgebung zu erzeugen.

Acknowledgments

I thank Prof. Andreas Zell for supervising and funding this dissertation project. I greatly appreciate the excellent working conditions and trust I enjoyed that let me relatively freely choose my topics of interests and pursue my research throughout the years .

I am extraordinarily grateful to my colleagues who created a cordial and helpful work environment. I would like to thank especially Shaowu Yang, Konstantin Schauwecker, Daniel Dube, Artur Koch and Lixing Jiang for their always fruitful discussions and help, which led to several interesting joint projects and publications.

I thank my long-time office-roommates turned friends Artur Koch, Lixing Jiang, and Vo Duc My for all the friendly and entertaining coffee breaks.

I am deeply indebted to my wife Tinatin and my daughter Nino for their support and understanding, especially whenever there was an important deadline approaching: You kept me sane.

Finally, I would never have made it without the constant support and encouragement from my parents Eberhard and Elenore. I am heartbroken by the fact that neither lives to see my graduation. I thank you both with all my heart.

Contents

Chapter 1

Introduction

1.1 Motivation

Micro Aerial Vehicles (MAVs) are small, light-weight, and often inexpensive flying machines that can easily get to places difficult to access for humans. Their popularity is steadily growing, for example in agricultural applications monitoring the growth of crops as described in Herbst (2010), in military reconnaissance for example for the German Bundeswehr (see Bundeswehr (2013)), for firefighters measuring the pollutant concentration in the air (see Bandemann (2010)), and in search and rescue or disaster relief scenarios as in Fukushima (see Honig (2011)). There are also several industrial applications for MAVs already, mostly focusing on aerial photography, surveying and geomatics (e.g. Schällibaum AG (2014)), and most recently even product delivery: Within one year, several big companies announced they were backing ambitious projects to allow automatic delivery based on MAVs. Amazon started in December 2013 by announcing their plans for a future delivery service *Amazon Prime Air* at a rather early stage (see Amazon.com, Inc. (2013)). Within the same month, DHL showcased an MAV that was used to deliver medication across the river Rhine (see Lang (2013)), which was in the meantime extended to deliver medication to a pharmacy on the island Juist, 12 km off the north sea coast of Germany (see Hern (2014)). In the meantime, Google announced their own delivery system called *Project Wing*, that was able to deliver first aid kits, among other things, to farmers in Australia (see Rushe (2014)). The latest company to join the delivery efforts was France's La Poste, which announced that it is now also developing a Hexacopter to deliver medication to destinations as far as 19 km away (see Samuel (2014)).

Even though some examples above seem quite advanced already, they are still severely limited in their on-board sensing and intelligence capabilities: They are either remotely controlled by a skilled operator or rely on absolute position estimates obtained via GPS, which is only reliable in the open, away from buildings or trees. One important research topic in robotics is thus the development of *truly autonomous micro aerial vehicles* that can also operate in *GPS-denied environments*. While most of the MAVs mentioned above use only inertial sensors and GPS, truly autonomous MAVs require more on-board sensing and processing capabilities: They are required to be able to actually perceive their environment.

Early research relied on mounting light-weight laser range finders on MAVs. This was a logical first step, since laser range finders are known to enable robust autonomous navigation of wheeled mobile robots in unknown roughly planar (i.e. 2D) environments. The most important examples of this approach can be found in Grzonka *et al.* (2009), which describes the first MAV navigating autonomously indoors using a laser range finder assuming a 2D environment, Bachrach *et al.* (2011), where a similarly-equipped MAV was able to autonomously fly through openings resembling windows, and in Bry *et al.* (2012), where an autonomous fixed-wing aircraft was demonstrated flying aggressive maneuvers and localizing itself using a combination of laser range finder and inertial sensors within a previously mapped indoor environment.

Laser range finders are often heavy, expensive, and only provide range measurements within a 2D plane[1]. An alternative paradigm that became more and more popular during the recent years is relying on cameras for autonomous navigation. Compared to laser range finders, cameras are smaller, lighter, cheaper, and provide richer information about the environment. The downside is that image processing requires more computational power, which in turn necessitates stronger and thus often heavier on-board computers. Early experiments with MAVs capable of camera-based navigation relied on artificial markers such as infrared LEDs as in Wenzel and Zell (2009) or ARTags as in Meier *et al.* (2011). If MAVs should be able to navigate using natural landmarks alone, they need to employ methods of visual odometry or SLAM. The combination of a monocular visual SLAM system with inertial measurements to allow autonomous navigation was described in Weiss *et al.* (2011). A basic obstacle-avoidance and exploration scheme using stereo cameras was shown in Fraundorfer *et al.* (2012), and the first MAV using an RGBD camera for localization in Huang *et al.* (2011).

[1]Note that 3D laser range finders do exist, of course, but the models currently available are too heavy to be of much practical use for MAV research.

1.2 Contributions & Outline

This dissertation is focused on various aspects of efficient visual SLAM using RGBD cameras for use on micro aerial vehicles. The first chapters up to chapter 5 cover important foundations required for a proper understanding of the later technical chapters. The main contributions are split up into chapters in the following way:

The first chapters 2 and 3 recapitulate foundational knowledge about cameras and their mathematical models (chapter 2) and the foundations of simultaneous localization and mapping and its relation to maximum likelihood estimation and least squares (chapter 3), which is recommended for a proper understanding of the subsequent chapters.

Chapter 4 describes in detail the monocular visual SLAM system *Parallel Tracking and Mapping* (PTAM) presented in Klein and Murray (2007) and how it was adjusted to be used on our robots. This is the foundation for the later work especially in chapters 5 and 9. In conjunction with a very efficient sign-detection algorithm described in Scherer *et al.* (2011), this also contributed to the following co-authored publications:

- Yang, S., Scherer, S. A., and Zell, A. (2013a). An Onboard Monocular Vision System for Autonomous Takeoff, Hovering and Landing of a Micro Aerial Vehicle. *Journal of Intelligent & Robotic Systems*, **69**(1–4), 499–515

- Yang, S., Scherer, S. A., Schauwecker, K., and Zell, A. (2014a). Autonomous Landing of MAVs on Arbitrarily Textured Landing Sites using Onboard Monocular Vision. *Journal of Intelligent & Robotic Systems*, **74**(1-2), 27–43

In chapter 5 we present a novel method of integrating depth measurements in visual SLAM. The major findings in this chapter were previously published in the following conference paper:

- Scherer, S. A., Dube, D., and Zell, A. (2012). Using depth in visual simultaneous localisation and mapping. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 5216–5221, St. Paul, Minnesota, USA

Since this method obviously lends itself to being applied to depth values inferred from stereo matching, the following co-authored paper is also partially based on this work:

- Schauwecker, K., Ke, N. R., Scherer, S. A., and Zell, A. (2012a). Markerless Visual Control of a Quad-Rotor Micro Aerial Vehicle by Means of On-Board Stereo Processing. In *22nd Conference on Autonomous Mobile Systems (AMS)*, pages 11–20, Stuttgart, Germany. Springer

Chapter 6 describes the implementation of a full visual SLAM system that can serve as a replacement for PTAM using RGBD cameras. Its advantage over PTAM is the fact that it can detect and close loops in real-time on the computationally constrained on-board computer of our MAV. This system was previously published in the following conference paper:

- Scherer, S. A. and Zell, A. (2013). Efficient Onboard RGBD-SLAM for Fully Autonomous MAVs. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1062–1068, Tokyo Big Sight, Japan

In chapter 7 we investigate whether loop closure detection using depth images instead of grayscale intensity images is feasible. We compare the performance of various feature detection and description methods with regards to loop closure detection in comparison to the state of the art method for grayscale images. This work was previously published in the following conference paper:

- Scherer, S. A., Kloss, A., and Zell, A. (2013). Loop Closure Detection using Depth Images. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 100 – 106, Barcelona, Catalonia, Spain

Chapter 8 describes an efficient ground plane detection algorithm for depth images that is based on a novel inlier/outlier/worse-outlier model and how this can be used for correcting drift in visual odometry and SLAM. This was also published in the following conference paper:

- Scherer, S. A., Yang, S., and Zell, A. (2015). DCTAM: Drift-corrected tracking and mapping for autonomous micro aerial vehicles. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 1094–1101, Denver, CO, USA

Chapter 9 combines the individual benefits from chapters 5 to 8 and unites the tracking performance of PTAM with the advantages of a full SLAM back-end with loop closing and optionally integrating ground plane measurements to enable autonomous navigation of an MAV in GPS-denied environments. This is also described in the conference paper above. The methods described in this chapter, specifically improving PTAM by adding loop closure and pose graph optimization, further contributed to the following co-authored paper:

- Yang, S., Scherer, S. A., and Zell, A. (2014b). Robust onboard visual SLAM for autonomous MAVs. In *2014 International Conference on Intelligent Autonomous Systems (IAS-13)*, Padova, Italy

Finally, chapter 10 concludes this dissertation with a short summary and outlook towards future work.

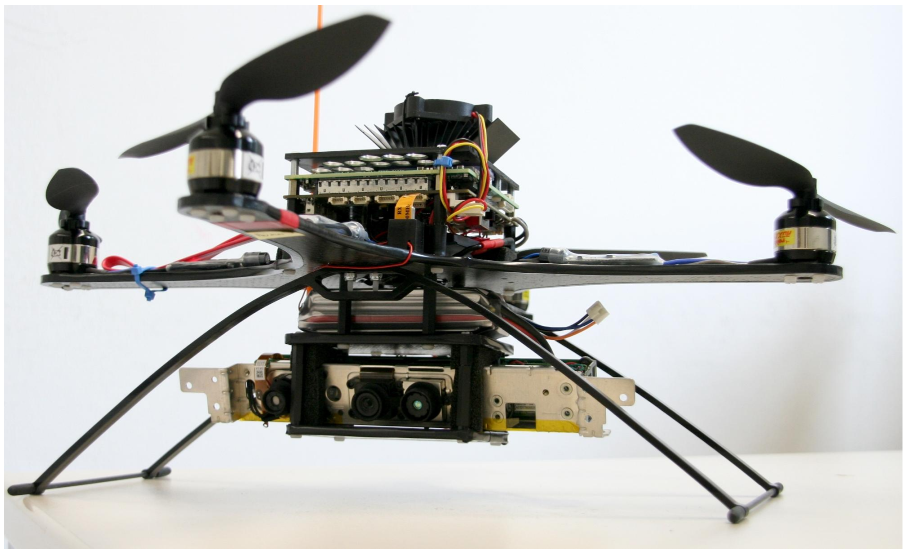


Figure 1.1: The initial version of our MAV.

1.3 Experimental Platform

1.3.1 Hardware

All MAV-related experiments throughout this work were performed with the same experimental platform that underwent several incremental updates during the course of this PhD project: A small quadrotor helicopter built to resemble the Pixhawk Cheetah open source MAV described in Meier *et al.* (2011), which in essence is a Mikrokopter *MK-Quadro*[2] quadrotor helicopter with a lighter frame, different onboard electronics, and an additional powerful on-board computer.

Our initial setup used the motors, brushless motor controllers (BLMC), propellers, and landing gear from the Mikrokopter Project. In order to save some weight, the Pixhawk project used their own frame consisting of a single piece of CFK sandwich plate. Since the original Mikrokopter FlightCtrl does not allow autonomous control,[3] the Pixhawk project replaced the original flight control unit (FCU) *Mikrokopter FlightCtrl* with an alternative FCU called *pxIMU* and added a custom-made COM express base board called *pxCOMex* that can host a very light-weight industrial computer-on-module (COM), in our case a Kontron microETXexpress®-PC SL9400, which includes an Intel Core 2 Duo CPU clocked at 1.86 GHz, and used 2 GB of RAM and a light-weight SSD. Both pxIMU and pxCOMex were developed by Christian Dobler as his master thesis (see Dobler (2009)).

For the scope of this dissertation, we rely on an RGBD camera as the main sensor. We

[2]`http://wiki.mikrokopter.de/MK-Quadro`

[3]The code running on the FlightCtrl can be replaced by custom code that allows automatic control. This approach was chosen for the Telekyb system described in Grabe *et al.* (2013).

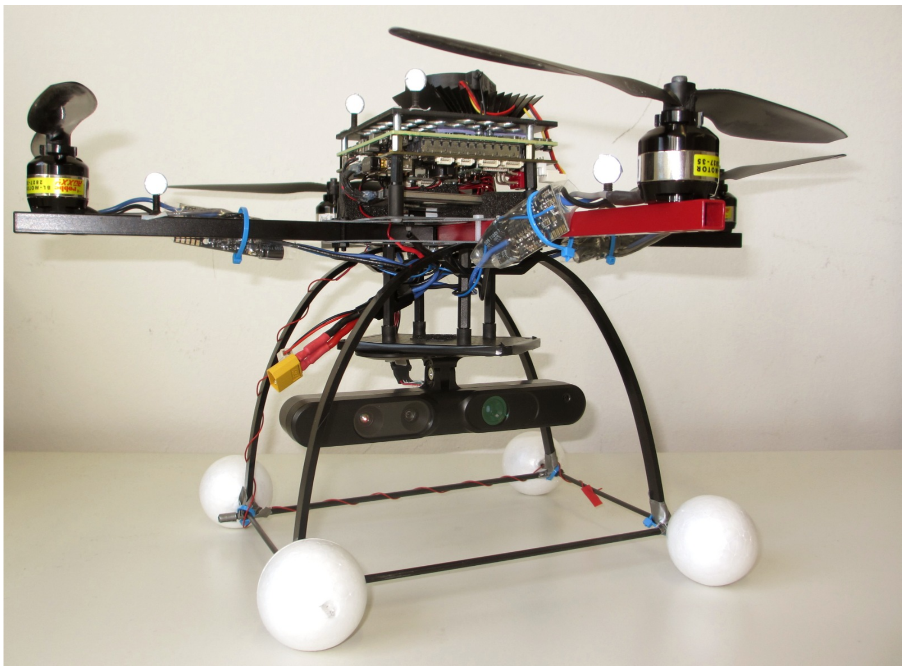


Figure 1.2: The latest version of our MAV.

also built other variants using one monocular camera as used in Masselli *et al.* (2014), two monocular cameras as used in Yang *et al.* (2014c) and two stereo cameras as used in Schauwecker and Zell (2014).

The initial version of our MAV is shown in figure 1.1. It is almost identical to the Pixhawk Cheetah quadrotor, except it uses a Microsoft Kinect RGBD camera that was stripped from its large encasing as its main sensor. The latest version of the MAV is shown in figure 1.2. It uses the more robust and cheaper but slightly heavier Mikrokopter frame. The Microsoft Kinect sensor was replaced by an ASUS Xtion Pro Live, which provides almost identical RGBD images at a much smaller size and lower weight.

1.3.2 Software

The on-board software that controls the MAV is divided into time-critical tasks that have to run in hard real time and are performed on the microcontroller on the one hand, and computationally more expensive tasks with no or only soft real-time constraints that are performed on the on-board computer.

The time-critical tasks include attitude estimation from measurements of the inertial measurement unit (IMU), position estimation from possibly noisy position estimates, position control, and attitude control.

All other computations are performed on the on-board computer. The most important ones are localization (in our case by means of visual SLAM) and waypoint navigation, i.e. determining the next desired position towards which the MAV should fly.

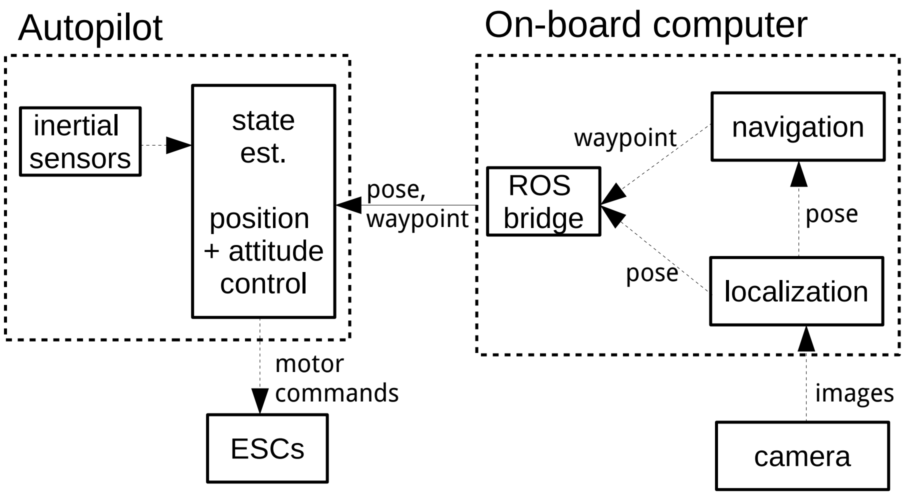


Figure 1.3: Illustration of the different hardware and software modules controlling the MAV.

The overall system is illustrated in figure 1.3: On the left hand side one can see the autopilot (which corresponds to the pxIMU flight control unit in our case) which is responsible for state estimation based on measurements of the inertial sensors (accelerometer, gyroscope and magnetometer) and control of the MAV by commanding motor velocities for the electronic speed controls (ESCs, i.e. the motor controllers). This part was implemented by the Pixhawk team already and we only performed minor modifications.

The on-board computer depicted on the right hand side communicates with the autopilot via a serial link connection using the so-called MAVLINK (see Meier (2009)) protocol. All software modules running on the on-board computer are implemented as nodes and nodelets for the robot operating system (ROS, see Quigley *et al.* (2009)), which among others allows convenient hardware abstraction, easy inter process communication, and easily reusing modules among different robot systems running ROS. There are always at least four nodes or nodelets running:

- The ROS bridge converts between ROS and MAVLINK messages. We implemented this ROS node to easily access and log all incoming messages from the autopilot and so we can provide the required pose estimates and waypoints as ROS instead of MAVLINK messages in the other modules.

- The navigation module decides about where the robot should fly. In the simplest case, it might always provide a constant position, which would lead to the MAV hovering in place. For waypoint-based path following, it also needs to know the current pose of the MAV.

- The camera driver receives the stream of images from the camera and makes it available through the ROS message infrastructure.
- Finally, the localization module is responsible for estimating the MAV pose, which is required for autonomous flight. This module might either use artificial markers for absolute localization with respect to these or natural landmarks for relative localization by means of visual odometry or SLAM.

The main focus of this dissertation is on this *localization* module above. It describes various alternatives and improvements to localization by means of visual SLAM. The formal limitations of this thesis unfortunately do not allow us to go into much detail about the physical properties and questions of estimation and control of the MAV required for an understanding of the inner workings of an autopilot. We have to treat it largely as a black box and refer the interested reader to the recent textbook Beard and McLain (2012) and the seminal survey paper Mahony *et al.* (2012) for more details.

Chapter 2

Cameras

2.1 Camera Models

Camera models describe the geometry of how objects in 3D space are projected into 2D images when captured by a camera. This task is so essential for computer vision that there is a chapter dedicated to it in almost every computer vision textbook (see e.g. Trucco and Verri (1998), Faugeras *et al.* (2001), Ma *et al.* (2003), Hartley and Zisserman (2004), and Szeliski (2010)).

In the following definitions, we slightly depart from the most popular notation using homogeneous coordinates in all steps in order to facilitate the computation of Jacobians required later (e.g. section 4.4.4).

2.1.1 Pinhole Camera Model

The simplest camera consists of a light-proof box with only one tiny opening ("pinhole"). The fact that light shining through the opening will project a mirrored image of the outside world onto an image plane was described as early as in ca. 330 BC by Aristotle and exploited in a description of Alhazen around 1020 AD. A brief history of the pinhole camera can be found in the first chapter of Renner (1995).

If we assume the pinhole to be infinitesimally small, all light rays entering the camera need to pass through the same focal point $\boldsymbol{O}$. A point in 3D space $\boldsymbol{p} = (p_x, p_y, p_z)^T$ will be projected through $\boldsymbol{O}$ onto the 2D point $\boldsymbol{u} = (u_x, u_y)^T$ on the image plane such that:

$$\boldsymbol{u} = \begin{pmatrix} u_x \\ u_y \end{pmatrix} = -f \begin{pmatrix} \frac{p_x}{p_z} \\ \frac{p_y}{p_z} \end{pmatrix} \tag{2.1}$$

Note the negative sign in equation (2.1): The projected image captured by a pinhole camera is reflected in $\boldsymbol{o}$. This is illustrated in figure 2.1.

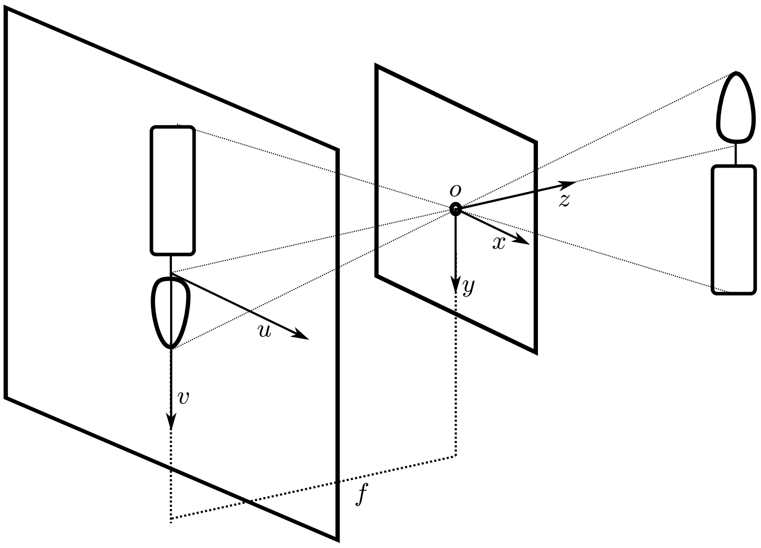


Figure 2.1: Illustration of the basic pinhole camera model. The actual object is shown on the right, the focal point is in $\boldsymbol{o} = (0,0,0)^T$, and the image plane on the left at focal length f away from the focal point. Note how the projection of the object appears upside down (mirrored).

2.1.2 Idealised Camera Model

In a physical camera, the image plane obviously has to be behind the focal point. If we could place the image plane such that it is exactly 1 unit of length *in front of the focal point*, which is equivalent to setting f in eq. (2.1) to $f = -1$, the projection above would be even easier:

$$\boldsymbol{n} = \begin{pmatrix} n_x \\ n_y \end{pmatrix} = n(\boldsymbol{p}) = \begin{pmatrix} \frac{p_x}{p_z} \\ \frac{p_y}{p_z} \end{pmatrix} \tag{2.2}$$

This is a perspective projection without any parameter. Point coordinates $\boldsymbol{n}$ projected according to eq.(2.2) are often called *normalized image coordinates*. They are especially useful when inverting the projection using known depth values, since:

$$\boldsymbol{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} n_x \\ n_y \\ 1 \end{pmatrix} d \tag{2.3}$$

Where $d = p_z$ is the depth of point $\boldsymbol{p}$. The homogeneous version $\boldsymbol{n}' = (n_x, n_y, 1)$ of $\boldsymbol{n}$ coincides with the direction of its corresponding ray. It means that any point on the line $(s \cdot \boldsymbol{n}')$ in 3D space is projected onto the same 2D point $\boldsymbol{n}$.

2.1.3 Linear Camera Model

When using images captured using digital cameras, we need to work with pixel coordinates to access intensities at certain image locations. In pixel coordinates, $x > 0$ and $y > 0$ denote the column and row of the cell within its pixel grid.

The most common transformation between normalized coordinates $\boldsymbol{n}$ and pixel coordinates $\boldsymbol{u}$ is the following affine transformation:

$$\boldsymbol{u} = u(\boldsymbol{n}) = \begin{pmatrix} f_x & \gamma \\ 0 & f_y \end{pmatrix} \cdot \boldsymbol{n} + \boldsymbol{c} = \begin{pmatrix} f_x \cdot n_x + \gamma \cdot n_y + c_x \\ f_y \cdot n_y + c_y \end{pmatrix} \tag{2.4}$$

The parameters f_x, f_y, γ, c_x and c_y are called intrinsic camera calibration parameters: They completely describe the linear camera model. The parameters $f_x = \frac{f}{s_x}$ and $f_y = \frac{f}{s_y}$ are the ratios of focal length and pixel size s_x and s_y in x and y dimension, $\boldsymbol{c} = (c_x, c_y)^T$ is the position of the principal point, i.e. the projection of the focal point $\boldsymbol{o}$ onto the image plane, and γ is a skew coefficient which depends on the angle between pixel columns and rows, but is typically 0.

This model is called the linear camera model even though equation (2.4) is in fact not linear but affine. But it can be expressed as a linear equation using homogeneous coordinates and a camera matrix $\boldsymbol{K}$ which contains all intrinsic parameters:

$$\begin{pmatrix} u_x \\ u_y \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{\boldsymbol{K}} \cdot \begin{pmatrix} n_x \\ n_y \\ 1 \end{pmatrix} \tag{2.5}$$

2.1.4 Non-Linear Distortions

Real lenses often exhibit non-linear distortions which cannot be described using a linear model alone. These distortions are especially prominent for lenses with wide field of view or short focal length. An example of this effect is shown in figure 2.2. The linear camera model dictates that projections of straight lines also have to be straight. This is obviously not the case in this example where straight walls of buildings appear bent in the camera image.

The linear camera model shown above is usually extended to accommodate non-linear distortions by applying a non-linear distortion function to the normalized image coordinates before they are transformed by the affine transformation of the linear model in equation (2.5). Various distortion models exist, but we will only describe the two that are most relevant for this work.

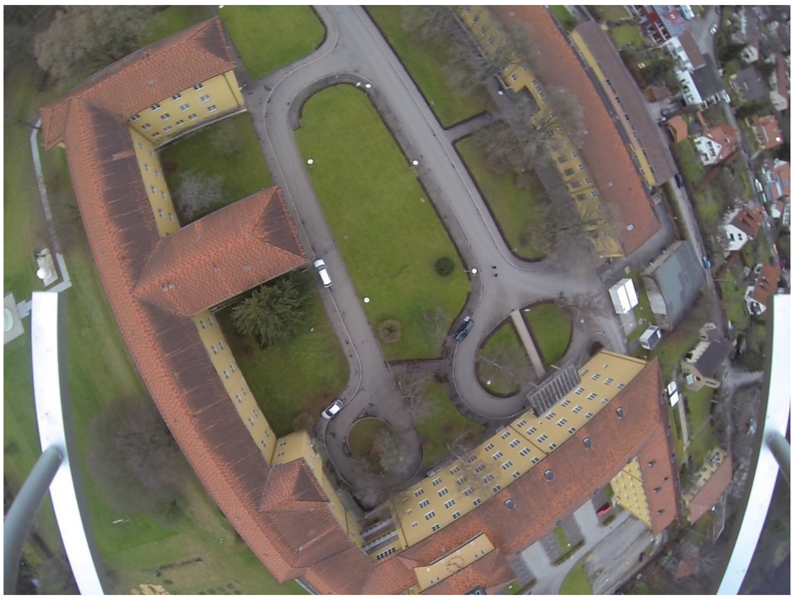


Figure 2.2: Example image taken with a camera exhibiting considerable lens distortion. Photograph used with permission by Norbert Morgenstern.

FOV or ATAN Model

The FOV (field of view) camera model by Devernay and Faugeras (2001), sometimes also called ATAN camera model, can model strong radial distortion typically found in fish-eye lenses. It computes the distorted normalized coordinates $\boldsymbol{d}$ using the following formula:

$$\boldsymbol{d} = d(\boldsymbol{n}) = \begin{pmatrix} d_x \\ d_y \end{pmatrix} = \frac{r_d}{r_u} \begin{pmatrix} n_x \\ n_y \end{pmatrix} \tag{2.6}$$

Where $r_u = \sqrt{n_x^2 + n_y^2}$ is the undistorted radius. The distorted radius r_d is computed as:

$$r_d = \frac{1}{\omega} \arctan\left(2r_u \cdot \tan\frac{\omega}{2}\right) \tag{2.7}$$

Here ω is the single distortion parameter and represents the field of view of a corresponding ideal fish-eye lense. This distortion model can also easily be inverted, which is required for undistorting an image.

$$r_u = \frac{\tan(\omega \cdot r_d)}{2\tan\frac{\omega}{2}} \tag{2.8}$$

The FOV model is the one originally used in Klein and Murray (2007), likely because it requires only one additional parameter, is suitable for fish-eye lenses with a wide field of

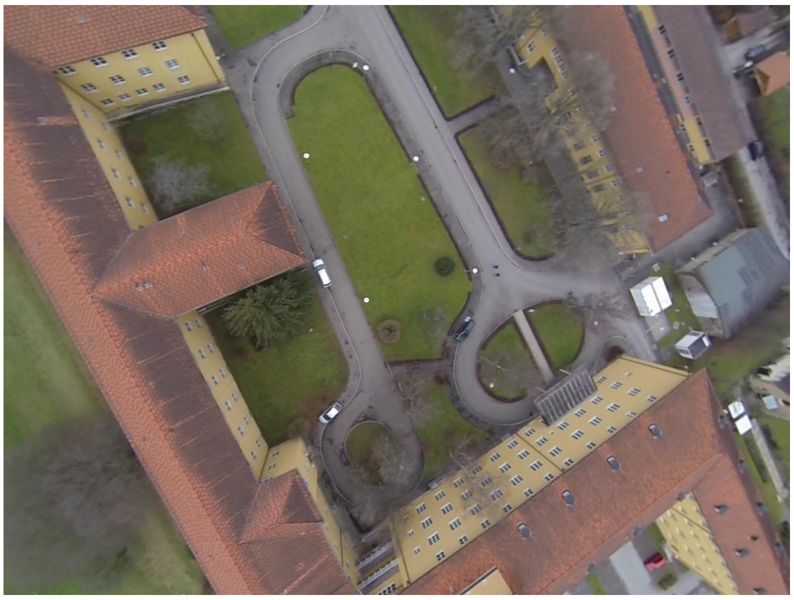


Figure 2.3: An undistorted version of the example image from figure 2.2.

view, is easily inverted, and because the derivatives are easy to compute analytically.

Polynomial Model

The most widely-adopted distortion model for regular lenses is the polynomial distortion model.

$$\boldsymbol{d} = d(\boldsymbol{n}) = \begin{pmatrix} d_x \\ d_y \end{pmatrix} \tag{2.9}$$

$$= \underbrace{\left(1 + k_1 r_u^2 + k_2 r_u^4 + k_3 r_u^6\right) \begin{pmatrix} n_x \\ n_y \end{pmatrix}}_{\text{radial}} + \underbrace{\begin{pmatrix} 2p_1 n_x n_y + p_2(r^2 + 2n_x^2) \\ p_1(r^2 + 2n_y^2) + 2p_2 n_x n_y \end{pmatrix}}_{\text{tangential}} \tag{2.10}$$

With the undistorted radius r_u defined as before. It consists of both a radial and tangential distortion component with a total of 5 parameters $(k_1, k_2, k_3, p_1, p_2)$, but the first two parameters of the radial component dominate all others for most lenses, such that k_3, p_1, p_2 can often be assumed to be 0.

One problem when using the polynomial model is the fact that there is no analytical inverse of equation (2.10). It can be approximated using iterative numerical optimization, which is too slow for real-time operation. A common solution to this problem is building a lookup-table for the inverse function once in a preprocessing step.

2.1.5 Combining all Parts for the Full Model

Using the building blocks described so far, the full projection model h from a 3D position in space to a 2D pixel position of a digital camera can now be described by a combination of ideal projection n to normalized coordinates, distortion d to distorted normalized coordinates, and finally the affine transformation u to actual pixel coordinates:

$$\boldsymbol{u} = h(\boldsymbol{p}) = u\left(d\left(n\left(\boldsymbol{p}\right)\right)\right) \tag{2.11}$$

We can also rearrange these function blocks in order to produce an undistorted version of a distorted image. For each pixel position in the undistorted image $\boldsymbol{u}_u$ we can compute its corresponding position in the distorted image $\boldsymbol{u}_d$ and vice versa:

$$\boldsymbol{u}_u = u\left(d^{-1}\left(u^{-1}\left(\boldsymbol{u}_d\right)\right)\right) \tag{2.12}$$

$$\boldsymbol{u}_d = u\left(d\left(u^{-1}\left(\boldsymbol{u}_u\right)\right)\right) \tag{2.13}$$

The image in figure 2.3 for example was produced using equation (2.13): For each pixel in the new undistorted image, we can compute its location in the original image. Intensity values of pixels within the undistorted image can then be computed by interpolating the intensity at its corresponding location within the original distorted image.

2.2 Depth Cameras

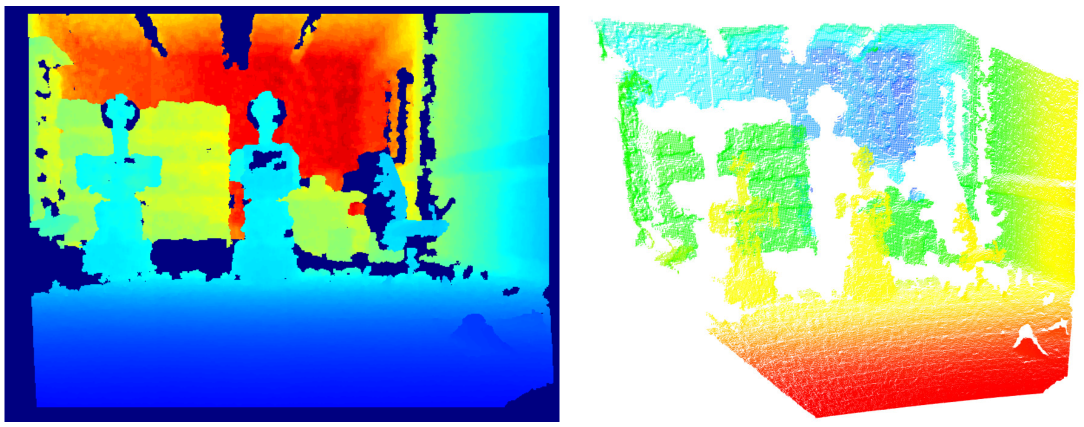
Depth cameras produce images which contain depth instead of intensity values for each pixel location. An example is shown in figure 2.4a: Low depth values are colored in blue, high values in red. Knowing the exact depth at each pixel location in conjunction with the camera calibration (i.e. the normalized image coorindates of each pixel) allows convenient reconstruction of a full point cloud using only one scalar-vector multiplication per pixel using equation (2.3)

Depth images thus provide rich information about the geometry of a scene, which can clearly be seen by the level of detail of the point cloud in figure 2.4b, which was reconstructed from the depth image in figure 2.4a.

2.2.1 Early Depth Cameras

Stereo Cameras

Stereo cameras may be considered the first depth cameras. Automatically inferring depth by matching areas between a pair of stereo images, i.e. two images of the same scene taken from slightly different viewpoints, has been studied since the seventies (see Hannah (1974)). A good survey of the earliest efforts in automatic stereo vision can be found in Barnard and Fischler (1982). Stereo vision remains an active field of research



(a) Depth image (b) Reconstructed point cloud

Figure 2.4: Depth image and reconstructed point cloud of an example scene.

until today, with the current work focusing either on optimizing accuracy, ideally on established benchmark datasets like the middlebury dataset described in Scharstein and Szeliski (2002) or the KITTY dataset described in Geiger *et al.* (2013), or optimizing the computation time required for specific applications as in Schauwecker *et al.* (2012b).

Problems when using stereo cameras are the still relatively high computational cost when computing dense depth images and its reliance on texture: Any stereo algorithm will inevitably fail for texture-less images.

Time-of-Flight Cameras

As opposed to stereo cameras, time-of-flight cameras are active sensors: They contain an emitter, typically an array of diodes, that emits infrared light invisible to the human eye, which is reflected by the scene and captured by the camera. Using special techniques to measure the time delay at which the signal arrives, it can infer the distance to visible objects.

According to Lange (2000), there are mainly two competing methods to measure this time delay:

- Pulsed modulation is the obvious approach: Light is emitted in short precisely timed pulses.

- With continuous modulation the emitted light is modulated by a known frequency much lower than the one of light itself. The time of flight can then be inferred from the phase shift between the emitted and received signal.

But continuous modulation seems to be becoming more and more popular in current sensors (e.g. Gokturk *et al.* (2004), Foix *et al.* (2011), and Mutto *et al.* (2012)).

Structured-Light Systems

Instead of modulating emitted light in the time domain as done by time-of-flight cameras, a scene can also be scanned using light modulated in the space domain. This idea is often attributed to Scharstein and Szeliski (2003), but it is in fact much older than that and was already described in Besl (1988). An easy-to-build and thus popular structured-light setup consists of a regular computer projector that projects several especially coded light patterns consisting of black and white stripes into the scene, which is picked up by a digital camera. This actively provides reliable dense stereo correspondences between camera and projector without the need for any natural texture.

The downside is that a single scan requires recording a multitude of different projected patterns, which makes this setup time-consuming and infeasible in dynamic environments.

2.2.2 Primesensor-Based Depth Cameras

Depth cameras finally had their breakthrough with the launch of Microsoft's Kinect sensor in November 2010, since this not only had a relatively high resolution for that time, but it was also really inexpensive. The depth sensing technique of the Kinect was developed by and licensed from Primesense, a small company bought by Apple in 2013. The original reference design by Primesense was called Primesensor and made available as a proof-of-concept sensor to prospective licensees and selected researchers only. Microsoft's Kinect was the first Primesensor-based RGBD camera available to the general public and was followed by several similar systems such as the Asus Xtion Pro (depth only) and Asus Xtion Pro Live (RGB and D).

Even though the exact depth sensing method employed by Primesense was never published in a scientific paper, the governing principle is described in full detail in the related patent application in Zalevsky *et al.* (2010):

The sensor projects a random but constant infrared laser speckle pattern into the scene which can be captured by an infrared camera. An example picture of this pattern is shown in figure 2.5a. An infrared camera mounted next to the sensor at a known distance (baseline) sees this pattern from a slightly different viewpoint. All parts of the pattern will thus be displaced depending on the relative scene depth at the location where they are reflected.

There is some confusion in the literature about the "true" resolution of the depth image. We know from Andreas Reichinger's reconstruction (see figure 2.5b) that the binary speckle pattern is of resolution 633×495. Primesense in the patent application (see Zalevsky *et al.* (2010)) suggests that the resolution of the infrared sensor should be chosen such that each speckle corresponds to roughly 2×2 pixels. We also know that at least the RGB camera employs a sensor with resolution 1280×960[1]. It is thus likely that the

[1] see `http://msdn.microsoft.com/en-us/library/jj131033.aspx`

(a) Pattern as captured by a photo camera that does not block the infrared spectrum.
Photograph used with permission by Karl E. Wenzel.

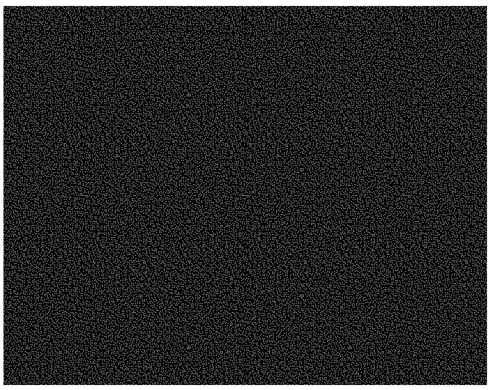

(b) Reconstruction of the binary pattern, see Reichinger (2011).
Picture used with permission by Andreas Reichinger.

Figure 2.5: Speckle pattern emitted by the infrared projector of Primesensor-based RGBD cameras.

sensor used in the infrared camera is the same. The chip on the sensor employs (accelerated) dense block matching over a certain matching window to estimate the optimal displacement and depth from the infrared image. The maximum resolution of the depth image actually returned by Primesensor-based depth cameras is VGA (640×480). One has to keep in mind, however, that each speckle and its displacement contribute to depth estimates of a whole region of neighboring depth estimates. Depth values of pixels close to each other are thus highly correlated and the information content of a depth image is thus lower than what the VGA resolution may suggest.

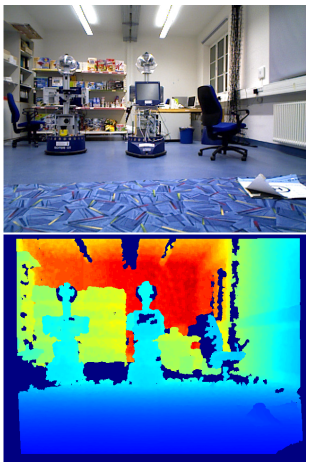
2.3 RGBD Sensors

Depth cameras alone already provide rich information about the environment, which in its nature is orthogonal to the data provided by traditional intensity- or color cameras. It is therefore only reasonable to try and combine both to build so-called RGBD cameras.

An ideal RGBD camera is a camera which can be described using the model introduced in sect. 2.1. For each pixel, however, it provides both color (RGB) and depth values. Even though such cameras typically provide two separate images for RGB and depth, we may think of depth as a fourth channel in one combined RGBD image.

2.3.1 Advantages of RGBD Images

The advantages of this representation are obvious: Given an RGBD image pair and the calibration parameters of the camera, we can very efficiently reconstruct a colored 3D


(a) Example RGBD image pair

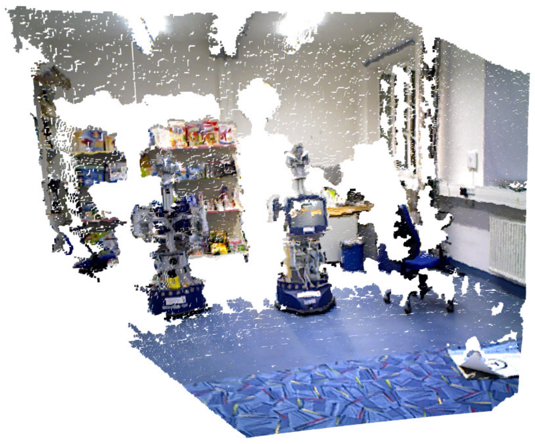

(b) Colored point cloud reconstructed from RGBD image and known intrinsic camera calibration.

Figure 2.6: Example RGBD image pair and reconstructed colored point cloud.

point cloud, again relying on equation (2.3). Even if we are only interested in colored point clouds from RGBD cameras, storing the original RGBD images requires less space than storing the full point cloud (one depth value instead of three coordinates per point): RGBD images are thus a more compact representation of point clouds obtained by RGBD cameras.

But even if we are not interested in reconstructing the full point cloud, we can directly access the depth value for any pixel in the RGB image, e.g. at locations of interest points, and compute the 3D position of the object to which this pixel belongs.

2.3.2 Obtaining RGBD Images from Real Data

In most cases, what we call an RGBD camera in fact consists of an RGB and a separate depth camera and their measurements need to be combined in order to compute a registered RGBD image. Two separate cameras cannot share the same origin but must be mounted next to each other with a relative pose ${}^{c_{RGB}}\boldsymbol{T}_{c_D} \neq \boldsymbol{I}_4$ that contains at least some translational displacement. Once this relative pose is known by calibrating both sensors with respect to each other (e.g. as described in Zhang and Pless (2004)), points of the original depth image can be warped to the RGB image's point of view and calibration.

$$\boldsymbol{u}_{RGB} = u_{RGB}(d_{RGB}(n(\boldsymbol{p}_{RGB}))) \tag{2.14}$$

$$\boldsymbol{p}_{RGB} = {}^{c_{RGB}}\boldsymbol{T}_{c_D} \cdot d_D^{-1}(u_D^{-1}(\boldsymbol{u}_D)) \tag{2.15}$$

We may now extend the measurement model of a regular camera defined in equation (2.11) to the one of an RGBD camera, in which any point $\boldsymbol{p} \in \mathbb{R}^3$ within the camera coordinate system is projected to pixel location $\boldsymbol{u}$ and might lead to a depth measurement d:

$$\begin{aligned} \boldsymbol{u} &= h(\boldsymbol{p}) = u\left(d\left(n\left(\boldsymbol{p}\right)\right)\right) \\ d &= d(\boldsymbol{p}) = (\boldsymbol{p})_z \end{aligned} \tag{2.16}$$

2.3.3 History of RGBD Cameras

Stereo cameras in combination with dense matching for depth estimation may be considered the first RGBD cameras. They are, however, usually not referred to as that. This is likely for several reasons: Since depth estimates from stereo vision heavily rely on texture, they are not as reliable compared to active depth cameras. The two measurement modalities of RGB and depth are not orthogonal but instead highly correlated in this case. Finally, dense stereo matching is computationally expensive, whereas so-called RGBD cameras typically provide the registered RGB and depth images without requiring any additional computations, with the depth image warped according to equation (2.15) in hardware already.

There were already some efforts to combine color-and depth cameras before the release of Primesensor-based RGBD cameras. These early approaches relied on early time-of-flight cameras, whose sensors were of very limited resolution. Fusing low-resolution depth images with high-resolution RGB images was thus an important research topic (see e.g. Huhle *et al.* (2007)).

In contrast to these early RGBD cameras mentioned above, modern RGBD cameras provide relatively high-resolution RGB and depth images and can warp the depth image to be aligned with its RGB image in hardware already. This frees up the CPU for other tasks.

Chapter 3

Mathematical Foundations of Simultaneous Localization and Mapping

3.1 Maximum Likelihood Estimation

A constantly recurring pattern in robotics is finding parameters of a model that agree best with noisy measurements of various sensors. Prominent examples are state estimation using Kalman Filters described in Kalman (1960), which can be considered a recursive maximum likelihood estimator of the system state (see Thacker and Lacey (1996)), occupancy grid mapping using forward sensor models as proposed in Thrun (2003), which tries to find the best assignment of either free or occupied attributes to grid cells such that it maximizes the joint likelihood of all measurements, and finally simultaneous localization and mapping, whose relation to maximum likelihood estimation will become clear later within this section.

3.1.1 Elementary Maximum Likelihood Estimation

A sensor in general reports multiple measurements $\boldsymbol{y} = (y_1, \dots y_n)^T \in \mathbb{R}^n$ which can depend on a wide range of factors: Its location, environment and possibly many more. We usually try to describe and include the most relevant dependencies in terms of a measurement function, also called the measurement model. One example of such a measurement model is equation (2.11), which describes how a distinct point in 3D space is measured in pixel location by a digital camera. A model may contain various unknown or uncertain parameters $\boldsymbol{x} = (x_1, \dots x_m) \in \mathbb{R}^m$, e.g. the 3D position of a point seen in the image, or the camera pose. Also, since measurements are in general subject to noise, we usually try to model the full probability distribution of sensor measurements given perfect knowledge of all model parameters:

$$p(\boldsymbol{y}|\boldsymbol{x}) \tag{3.1}$$

For many sensors, this is a joint probability of multiple individual measurements which are not directly correlated:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}) \tag{3.2}$$

The term above is a conditional probability distribution over $\boldsymbol{y}$, but the actually measured values y_i are known and thus constant, whereas $\boldsymbol{x}$ is unknown. We can consider the above as a function of $\boldsymbol{x}$ and call it the likelihood function $L_{\boldsymbol{y}}(\boldsymbol{x})$:

$$L_{\boldsymbol{y}}(\boldsymbol{x}) = p(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}) \tag{3.3}$$

The term likelihood here stems from the fact that it is a conditional probability conditioned on the variable as in the likelihood term of Bayes' rule:

$$\underbrace{p(x|y)}_{\text{posterior}} = \frac{\overbrace{p(y|x)}^{\text{likelihood}} \cdot \overbrace{p(x)}^{\text{prior}}}{\underbrace{p(y)}_{\text{normalization}}} \tag{3.4}$$

In robotics, "likelihood" is a slightly ambiguous term often used informally for identities that are similar to probabilities but do not agree with their strict definition (e.g. in the case above with the fact that probabilities need to be normalized). Examples can be found, among others, in ch. 6.4 of Thrun *et al.* (2005) and Olson (2009). The maximum likelihood estimate $\boldsymbol{x}_{mle}$ is the set of values for $\boldsymbol{x}$ that maximizes the above likelihood function:

$$\boldsymbol{x}_{mle} = \arg\max_{\boldsymbol{x}} L_{\boldsymbol{y}}(\boldsymbol{x}) = \arg\max_{\boldsymbol{x}} \prod_{i=1}^{N} p(y_i|\boldsymbol{x}) \tag{3.5}$$

3.1.2 MLE with Normal Distributions: Least Squares Minimization

If we assume each measurement to follow a normal distribution, we can write down the actual formula of $p(y_i|\boldsymbol{x})$:

$$p(y_i|\boldsymbol{x}) = \frac{1}{\sqrt{2\pi}\cdot\sigma} \exp\left(-\frac{(y_i - f_i(\boldsymbol{x}))^2}{2\sigma^2}\right) \tag{3.6}$$

The mean $f_i(\boldsymbol{x})$ is the expected measurement according to an ideal deterministic model given the true parameters $\boldsymbol{x}$. We can equivalently minimize the logarithm of the likelihood, which leads to much simpler terms:

$$
\begin{aligned}
\boldsymbol{x}_{mle} &= \arg\min_{\boldsymbol{x}} \log L_{\boldsymbol{y}}(\boldsymbol{x}) && (3.7)\\
&= \arg\min_{\boldsymbol{x}} \log \prod_{i=1}^{N} p(y_i|\boldsymbol{x}) && (3.8)\\
&= \arg\min_{\boldsymbol{x}} \log \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\cdot\sigma} \exp\left(-\frac{(y_i - f_i(\boldsymbol{x}))^2}{2\sigma^2}\right) && (3.9)\\
&= \arg\min_{\boldsymbol{x}} \frac{1}{2\sigma} \sum_{i=1}^{N} [y_i - f_i(\boldsymbol{x})]^2 && (3.10)\\
&= \arg\min_{\boldsymbol{x}} \underbrace{\boldsymbol{r}^T \cdot \boldsymbol{r}}_{S} && (3.11)
\end{aligned}
$$

Which corresponds to least squares minimization of the measurement errors. In the last row, we combined all individual measurement errors in one residual vector $\boldsymbol{r}$. If the model $f_i(\boldsymbol{x})$ is linear in $\boldsymbol{x}$, there is a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ such that:

$$f(\boldsymbol{x}) = \boldsymbol{A}\cdot\boldsymbol{x} \qquad (3.12)$$

The residual vector is then:

$$\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{A}\cdot\boldsymbol{x} \quad \text{with} \quad r_i = y_i - \sum_{j=1}^{m} A_{ij} x_j \qquad (3.13)$$

At the minimum of equation (3.11), the gradient of $S = \boldsymbol{r}^T\boldsymbol{r} = \sum_{i=1}^{n} r_i^2$ needs to be zero:

$$
\begin{aligned}
0 &= \frac{\partial S}{\partial x_j} && (3.14)\\
&= 2\sum_{i=1}^{n} r_i \frac{\partial r_i}{\partial x_j} && (3.15)\\
&= 2\sum_{i=1}^{n} \left(y_i - \sum_{k=1}^{m} A_{ik} x_k\right)(-A_{ij}) && (3.16)
\end{aligned}
$$

By rearranging terms it follows that:

$$\sum_{i=1}^{n} A_{ij} y_i = \sum_{i=1}^{n}\sum_{k=1}^{m} A_{ij} A_{ik} x_k \qquad (3.17)$$

If written in matrix-form this leads to the so-called normal equations:

$$\left(\boldsymbol{A}^T\boldsymbol{A}\right)\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{y} \tag{3.18}$$

This can be solved for $\boldsymbol{x}$ most efficiently by using the Cholesky decomposition if $\boldsymbol{A}^T\boldsymbol{A}$ is well-conditioned and positive definite or alternatively using the computationally more expensive singular value decomposition, even if it does not have full rank. (See Theorem 1.2.10 in Björck (1996))

3.1.3 Heteroskedasticity and Weighted Least Squares

In the previous section, we assumed all measurements to be normally distributed with equal variance σ^2. The covariance matrix of the vector of all measurements combined was a trivial diagonal matrix.

$$\boldsymbol{\Sigma} = \sigma^2\boldsymbol{I} \tag{3.19}$$

When combining multiple measurements, possibly obtained from different sensors, this is often not the case. In practice, the variance might depend on many factors and can be different for each measurement, i.e. measurements are heteroscedastic.

$$\boldsymbol{\Sigma} = diag(\sigma_1^2, \ldots \sigma_n^2) \tag{3.20}$$

The maximum likelihood estimate has to be computed as:

$$\boldsymbol{x}_{mle} = \arg\min_{\boldsymbol{x}} \frac{1}{2}\sum_{i=1}^{N}\left[\frac{y_i - f_i(\boldsymbol{x})}{\sigma_i}\right]^2 \tag{3.21}$$

$$= \arg\min_{\boldsymbol{x}} \underbrace{\boldsymbol{r}^T\boldsymbol{W}\boldsymbol{r}}_{S} \tag{3.22}$$

Notice the additional weight matrix $\boldsymbol{W}$ in the equation above, which combines information about all standard deviations σ_i. By comparing equations (3.21), (3.22), and (3.10), (3.11), we can see that the weight matrix has to be diagonal with:

$$\boldsymbol{W} = diag\left(\frac{1}{\sigma_1^2}, \ldots \frac{1}{\sigma_n^2}\right) \tag{3.23}$$

This weighted least squares system can be optimized by solving the following modified normal equations:

$$\left(\boldsymbol{A}^T\boldsymbol{W}\boldsymbol{A}\right)\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{W}\boldsymbol{y} \tag{3.24}$$

A more detailed introduction to weighted least squares can be found in chapter 5.5 of Montgomery *et al.* (2012).

3.1.4 Robust Least Squares

In real-world applications, using least squares as described before can lead to serious problems because it relies on all measurement noise to be Gaussian, which is rarely the case: The probability of measuring a value 6 standard deviations off the mean should be ca. 1 in one billion ($1:10^9$) assuming Gaussian noise. Practical experience tells us that completely unexpected values are much more likely than that, be it because of sensor failure, completely unexpected and thus not modeled effects, or sensor noise just not following a Gaussian distribution at all. According to Huber in Huber (1972) after all, "one never has a very accurate knowledge of the true underlying distribution".

The problem with the assumption of Gaussian noise is that one such theoretically very unlikely measurement will have huge a influence proportional to the square of its Mahalanobis distance and is enough to completely distort the result of least squares estimation. One common technique to overcome this problem are M-Estimators, introduced by Huber and first described in Huber (1964).

The main idea is to generalize least squares estimation: Instead of minimizing the square of the residuals $\sum_{i=1}^{N} r_i^2$, we can minimize a different function of the residuals

$$S = \sum_{i=1}^{N} \rho(r_i) \tag{3.25}$$

where ρ should be symmetric, positive-definite with a unique minimum at zero (see Zhang *et al.* (1997)). The minimum of this new objective function can be found in the same way as before:

$$\begin{aligned} 0 &= \frac{\partial S}{\partial x_j} && (3.26) \\ &= \sum_{i=1}^{n} \frac{\partial \rho(r_i)}{\partial r_i} \cdot \frac{\partial r_i}{\partial x_j} && (3.27) \\ &= \sum_{i=1}^{n} \psi(r_i) \cdot \frac{\partial r_i}{\partial x_j} && (3.28) \\ &= \sum_{i=1}^{n} w(r_i) \cdot r_i \cdot \frac{\partial r_i}{\partial x_j} && (3.29) \end{aligned}$$

With the influence function $\psi(x) = \frac{\partial \rho(x)}{\partial x}$ and the weight function $w(x) = \frac{\psi(x)}{x}$.

If we compare equation (3.29) to (3.15), it becomes clear that they are equivalent except for a new individual scaling factor for each summand. We can minimize the general objective function (3.25) by solving the following weighted least squares problem:

$$\arg\min_{\boldsymbol{x}} S = \arg\min_{\boldsymbol{x}} \sum_{i=1}^{N} \rho(r_i) = \arg\min_{\boldsymbol{x}} \sum_{i=1}^{N} w(r_i) r_i^2 \tag{3.30}$$

There are various popular choices for ρ and the corresponding weight function $w(r_i)$. Many choices of ρ turn robust least squares into maximum likelihood estimation assuming a non-Gaussian error distribution. Other choices do not correspond to any real distribution, but are very robust to outliers. Good overviews over popular options for ρ can be found in Zhang *et al.* (1997) and Hartley and Zisserman (2004) (A6.8, p.617). We illustrate some relevant examples in figure 3.1:

The most obvious choice is, of course, $\rho(x) = \frac{x^2}{2}$, which again results in regular least squares, i.e. maximum likelihood estimation for normally distributed errors. Another interesting choice is $\rho(x) = |x|$. This corresponds to least absolute difference minimization, which is a maximum likelihood estimator for errors following a Laplace distribution. We can see that ρ grows linearly with the error instead of quadratically, which corresponds to a probability distribution with heavier tails compared to the normal distribution. The Cauchy estimator is a maximum likelihood estimator assuming the errors follow a Cauchy distribution. Note that mean and variance of a Cauchy distribution are undefined. Its scale parameter in figure 3.1 was chosen to make it comparable to the other distributions. Our final example is the Tukey estimator, since it limits ρ to the constant value of an upper bound for measurements far away from the expected value: This effectively suppresses outliers, since $w(x) = 0$ for $|x| > c$. Because of this behavior, however, there is no corresponding probability distribution: Its probability density would have to converge to a certain value $p_\infty > 0$ for $x \to \pm\infty$, which means it cannot be normalized.

Iteratively Reweighted Robust Least Squares Algorithm

The problem with robust least squares is that we need an initial estimate of the model already, before we can compute errors and by extension weights w, which are in turn used to estimate the model. As opposed to normal least squares, we thus need to compute the maximum likelihood estimate in an iterative scheme:

Example Problem

An example in which we apply robust least squares to a toy problem is shown in figure 3.2: We want to fit a line model $y = mx + b$ to noisy measurements y_i at known locations x_i. The measurement noise of y_i in this case is not Gaussian but sampled from a Cauchy distribution, which is why we see more distant errors than expected with Gaussian noise.

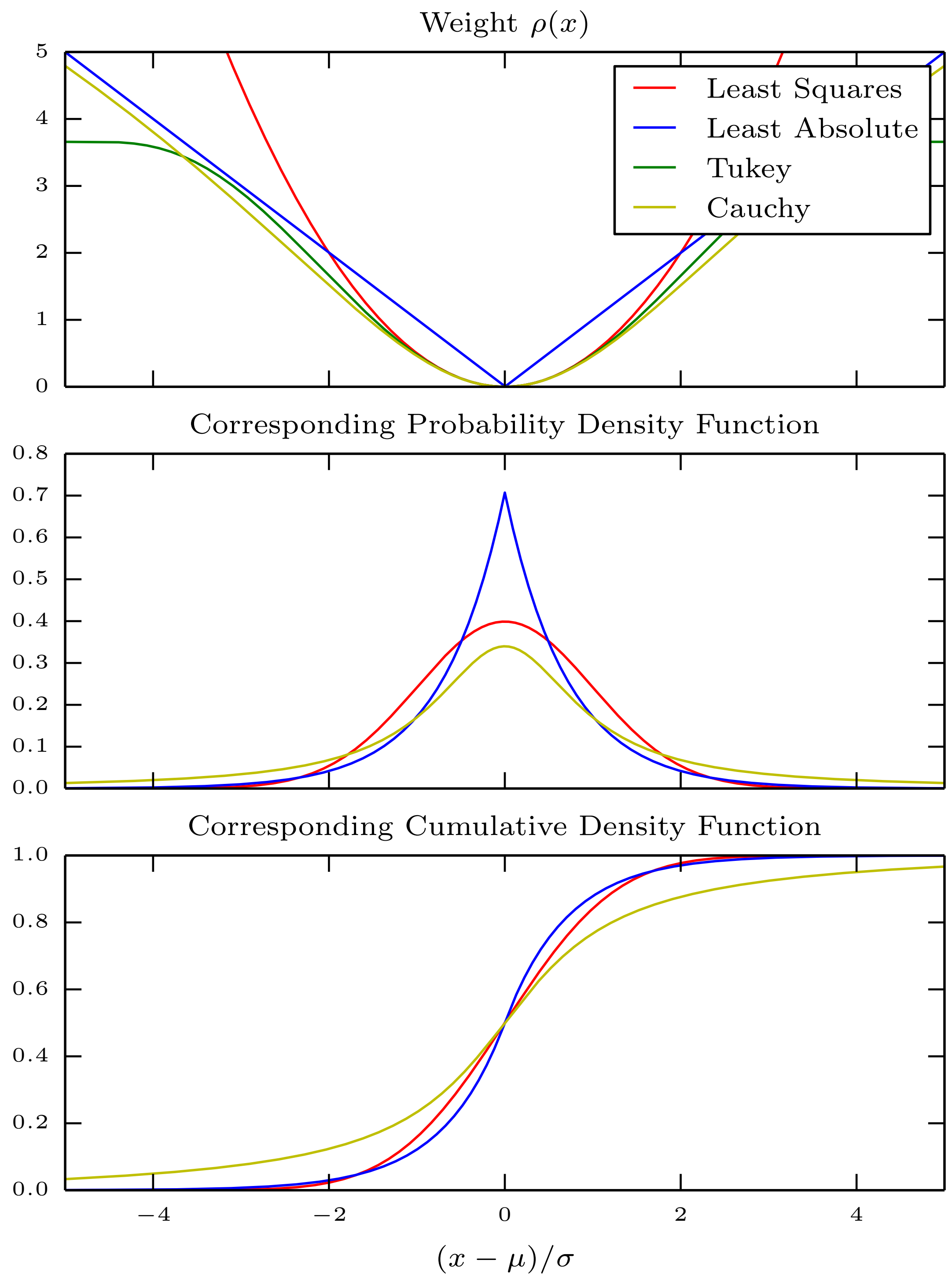


Figure 3.1: Comparison of different ρ-functions and their corresponding distributions.

Algorithm 1 Robust Least Squares

```
input: measurements $y_i$
$\boldsymbol{x}_{mle}$ := initial estimate or least squares solution
while not converged do
    compute weights $w_i$ based on $y$ and $\boldsymbol{x}_{mle}$
    compute $x_{mle}$ by solving weighted normal equations
end while
output: robust estimate $\boldsymbol{x}_{mle}$
```

We try to fit the best line using least squares (MLE assuming Gaussian noise), using reweighted least squares using Cauchy's weight function, and reweighted least squares with Tukey's biweight function.

We can clearly see from the top of figure 3.2 that the result of least squares is not a good estimate in this case and far from the true underlying model. Both robust estimates are much closer, with Tukey's weight function performing even better even though the result cannot be interpreted as a maximum likelihood estimate. We initialize both robust methods with all weights set to $w_i = 1 \;\; \forall i$, which is identical to the least squares estimate. The bottom of figure 3.2 shows how the Tukey estimate converges towards the true model with each iteration.

3.1.5 Non-Linear Least Squares

Thus far, we sustained the assumption that measurement models are linear, i.e. there was a matrix $\boldsymbol{A}$ such that $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$. In Robotics, this assumption is rarely valid and almost all practical models are non-linear, i.e. $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ for a non-linear function $\boldsymbol{f} : \mathbb{R}^m \mapsto \mathbb{R}^n$. We can still apply large parts of the methods described earlier after linearizing the non-linear model and residual at the latest estimate, which will lead to methods called non-linear least squares. (See Madsen *et al.* (2004) and Nocedal and Wright (2006) for thorough introductions.) The first-order Taylor series approximation of the residual vector $\boldsymbol{r}$ (see equation (3.13)) close to point $\boldsymbol{x}$ is:

$$\boldsymbol{r}(\boldsymbol{x}+\boldsymbol{h}) = \boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\boldsymbol{h} + O(||\boldsymbol{h}||^2) \tag{3.31}$$

$$\approx \boldsymbol{l}(\boldsymbol{h}) = \boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\boldsymbol{h} \tag{3.32}$$

The linearization $\boldsymbol{l}$ of $\boldsymbol{r}$ is a function of the deviation $\boldsymbol{h}$ instead of $\boldsymbol{x}$, since the latter remains fixed in this case. The last summand of equation (3.31) can be considered irrelevant for small $\boldsymbol{h}$ and is discarded for a first-order approximation. The matrix $\boldsymbol{J}(\boldsymbol{x}) \in \mathbb{R}^{n \times m}$ is the Jacobian matrix of $\boldsymbol{r}(\boldsymbol{x})$, i.e. the matrix consisting of first partial derivatives the vector result of residual $\boldsymbol{r}$ with respect to the model parameters $\boldsymbol{x}$:

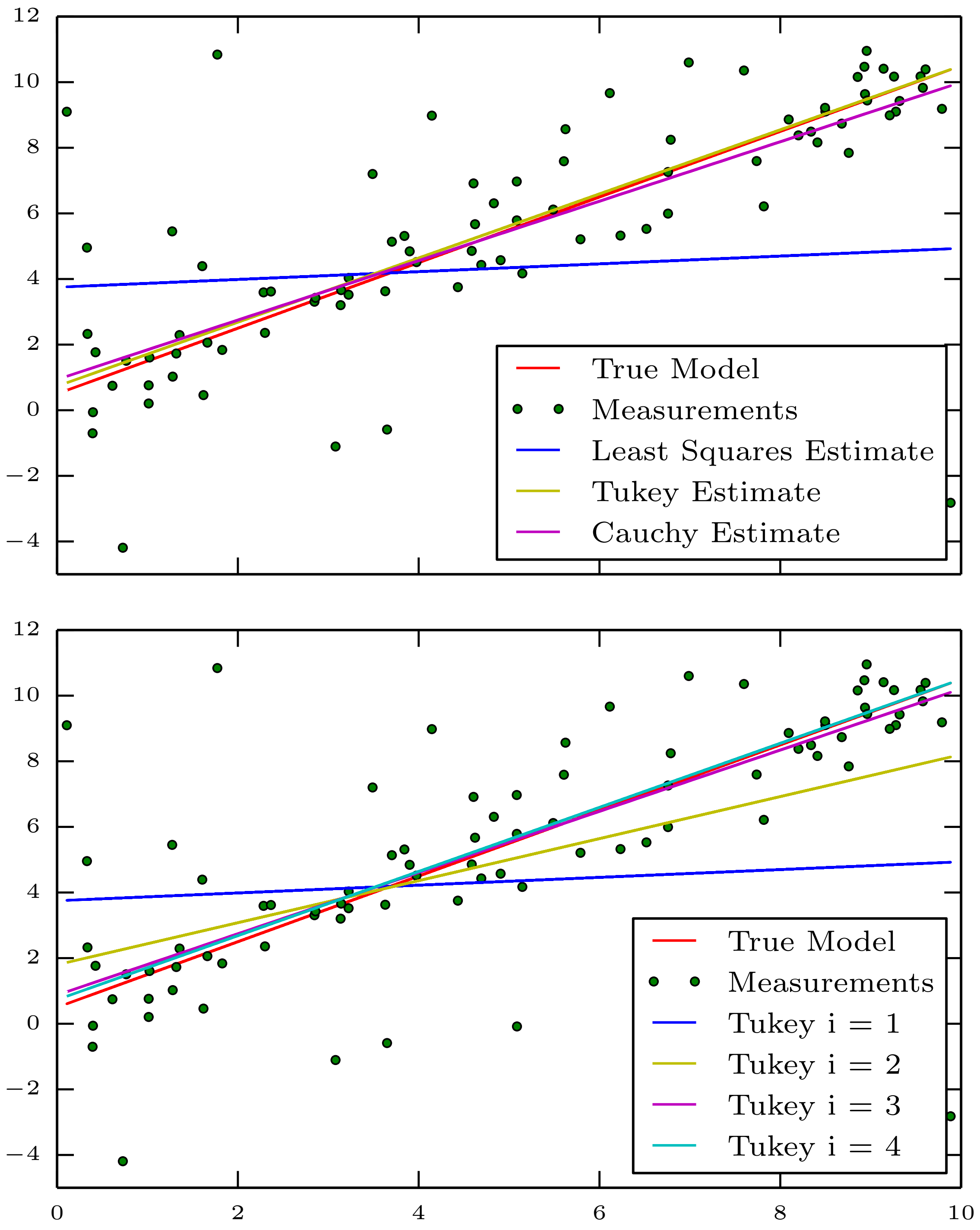


Figure 3.2: Line Fitting using Least Squares and Robust Least Squares

$$(\boldsymbol{J}(\boldsymbol{x}))_{ij} = \frac{\partial r_i}{\partial x_j}(\boldsymbol{x}) \tag{3.33}$$

As before, we want to minimize the squared residual, but our latest estimate $\boldsymbol{x}$ is now fixed, since this is the linearization point, and we are instead looking for the optimal step $\boldsymbol{h}$ away from $\boldsymbol{x}$:

$$\begin{aligned} L(\boldsymbol{h}) &= \frac{1}{2}\boldsymbol{r}(\boldsymbol{x}+\boldsymbol{h})^T\boldsymbol{r}(\boldsymbol{x}+\boldsymbol{h}) && (3.34)\\ &\approx \frac{1}{2}\boldsymbol{l}(\boldsymbol{h})^T\boldsymbol{l}(\boldsymbol{h}) && (3.35)\\ &= \frac{1}{2}\boldsymbol{r}^T\boldsymbol{r} + \boldsymbol{h}^T\boldsymbol{J}^T\boldsymbol{r} + \frac{1}{2}\boldsymbol{h}^T\boldsymbol{J}^T\boldsymbol{J}\boldsymbol{h} && (3.36)\end{aligned}$$

From equation (3.36) on, we implicitly assume $\boldsymbol{r} = \boldsymbol{r}(\boldsymbol{x})$ and $\boldsymbol{J} = \boldsymbol{J}(\boldsymbol{x})$ unless a different argument is explicitly provided. For L to be minimal its gradient $\boldsymbol{L}'(h)$ has to be 0:

$$\boldsymbol{L}'(h) = \frac{\partial L}{\partial \boldsymbol{h}} = \boldsymbol{J}^T\boldsymbol{r} + \boldsymbol{J}^T\boldsymbol{J}\boldsymbol{h} \overset{!}{=} 0 \tag{3.37}$$

This leads to the normal equations for non-linear least squares. They are almost identical to the linear case in equation (3.18) but instead of the data matrix $\boldsymbol{A}$ rely on the Jacobian $\boldsymbol{J}$, which may be considered the data matrix of the linearized model:

$$(\boldsymbol{J}^T\boldsymbol{J})\boldsymbol{h} = -\boldsymbol{J}^T\boldsymbol{r} \tag{3.38}$$

This is the core of each iteration of the Gauss-Newton Method for Non-Linear Least Squares:

Algorithm 2 Gauss-Newton Method for Non-Linear Least Squares

```
1: input: measurements $\boldsymbol{y}$
2: $\boldsymbol{x}$ := initial estimate
3: while not converged do
4:     compute Jacobian $\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}$ evaluated at current estimate $\boldsymbol{x}$
5:     compute step $\boldsymbol{h}$ by solving $(\boldsymbol{J}^T\boldsymbol{J})\boldsymbol{h} = -\boldsymbol{J}^T\boldsymbol{r}$
6:     $\boldsymbol{x} := \boldsymbol{x} + \boldsymbol{h}$
7: end while
8: output: estimate $\boldsymbol{x}$
```

Levenberg-Marquardt Method

The Gauss-Newton Method converges quickly if initialized close to the optimum but convergence may suffer and even fail completely if initialized far away (see Madsen *et al.* (2004)). Levenberg (1944) and Marquardt (1963) suggested modifying the Gauss-Newton method by adding a damping term to its normal equations:

$$(\boldsymbol{J}^T\boldsymbol{J} + \lambda\boldsymbol{I})\boldsymbol{h}_{lm} = -\boldsymbol{J}^T\boldsymbol{r} \tag{3.39}$$

The damping parameter $\lambda > 0$ is chosen heuristically to balance between pure Gauss-Newton optimization for small values of λ and small steps towards the direction of steepest descent for large values of λ. It also enforces the positive definiteness of the system, which is required when applying the Cholesky decomposition.

More in-depth treatments of Levenberg-Marquardt and other algorithms can be found in several textbooks (e.g. Nocedal and Wright (2006), Madsen *et al.* (2004), or Press *et al.* (2007)).

3.1.6 Non-Linear Optimization on Manifolds

In previous descriptions, we assumed the model parameters $\boldsymbol{x}$ to be elements of an m-dimensional vector space, i.e. $\boldsymbol{x} \in \mathbb{R}^m$. This is unfortunately not the case for almost any problem in robotics and computer vision, since they often include orientations, which are elements of the special orthogonal Lie groups $\mathrm{SO}(2)$ or $\mathrm{SO}(3)$ (see Hall (2003)).

There are various means of representing elements of Lie groups as real vectors. Elements of $\mathrm{SO}(3)$ are e.g. often represented using all entries of the corresponding rotation matrix, Euler angles, normalized quaternions, or rotation axis and angle. Optimizing on their vector representation directly, however, is a bad idea since they are not globally Euclidean, which leads to several problems outlined in Blanco (2010): Analytical Jacobians might not exist, there are singularities (such as the so-called ”Gimbal Lock”), and non-minimal representations offer extra degrees of freedom, which can lead to the optimization getting stuck moving along redundant solutions.

It is much more elegant and convenient to optimize “on the manifold”, i.e. store elements using any non-minimal but accurate representation and approximate the manifold by the Euclidean space it resembles locally, close to a reasonable reference point (e.g. its latest estimate) for correction steps. Optimizing on the manifold has been rediscovered for robotics applications in recent years (e.g. in Hertzberg (2008), Grisetti *et al.* (2010a), Hertzberg *et al.* (2013)), even though it had been known for much longer (see Gabay (1982), Smith (1993), Taylor *et al.* (1994)) and was already applied extensively in Bundle Adjustment and several visual SLAM systems (e.g. Triggs *et al.* (2000) Klein and Murray (2007) Eade (2008)) before.

Lie Groups

Lie groups are mathematical groups and differentiable manifolds (see Lee (2003)). They are not globally Euclidean but resemble Euclidean space in the local neighborhood of each element. Lie groups typically encountered in robotics and computer vision are rotations in n-dimensional space (the special orthogonal group $\mathrm{SO}(n)$) and rigid body transforms consisting of both rotations and translations in n dimensions (the special euclidean group $\mathrm{SE}(n)$). Both are matrix groups, i.e. they can be represented using real matrices $\boldsymbol{M} \in \mathbb{R}^{m \times m}$ and their group operation corresponds to matrix multiplication. Even though elements of Lie groups are represented using real matrices in $\mathbb{R}^{m \times m}$, their degrees of freedom k are in general $k < m^2$ due to additional constraints. For rotation matrices, e.g., these are $\boldsymbol{M} \cdot \boldsymbol{M}^T = \boldsymbol{I}$ and $\det(\boldsymbol{M}) = 1 \; \forall \boldsymbol{M} \in \mathrm{SO}((n)))$

Lie Algebra

Each Lie group G with k degrees of freedom has an associated Lie algebra $\mathfrak{g}$, which is the k-dimensional tangent space of G around the identity. Its canonical basis $(\boldsymbol{G}_1, \cdots \boldsymbol{G}_k)$ is called the set of k generators of $\mathfrak{g}$. If the Lie group is a matrix group, then elements of the Lie algebra $\boldsymbol{e} \in \mathfrak{g}$ can also be represented as matrices $\in \mathbb{R}^{m \times m}$. It is often more convenient, however, to represent elements of $\mathfrak{g}$ using a linear combination of the k generators:

$$\boldsymbol{e} = \sum_{i=1}^{k} c_i \boldsymbol{G}_i \tag{3.40}$$

We will assume that elements of a Lie algebra are represented by their linear combination of generators $\boldsymbol{c} = (c_1, \cdots c_k)^T \in \mathbb{R}^k$ from now on, unless otherwise stated.

Exponential Map

A Lie group and its Lie algebra are connected by the exponential map which takes elements from the Lie algebra to the group: $\exp : \mathfrak{g} \mapsto G$. For matrix Lie groups, i.e. all groups we encounter in this work, this is the matrix exponential:

$$\exp x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i \tag{3.41}$$

Optimization on the Manifold

We can now use a generalized addition operator $\boxplus : G \times \mathbb{R}^k \mapsto G$ as proposed in Hertzberg (2008) and adopted in Blanco (2010) for corrections to Lie groups on the manifold:

$$\boldsymbol{x} = \boldsymbol{x} \boxplus \boldsymbol{\epsilon} \Longleftrightarrow \boldsymbol{x} = \exp(\boldsymbol{\epsilon}) \cdot \boldsymbol{x} \tag{3.42}$$

The state $\boldsymbol{x}$ might in fact be a product of mixed elements of Lie groups and real vector spaces. For factors that are not elements of Lie groups, $\boxplus$ should fall back to the basic addition operator. In order to find the optimal incremental update, we want the following slightly different Jacobian to be 0:

$$\boldsymbol{J}(\boldsymbol{x}) = \left. \frac{\partial \boldsymbol{r}(\boldsymbol{x} \boxplus \boldsymbol{\epsilon})}{\partial \boldsymbol{\epsilon}} \right|_{\boldsymbol{\epsilon}=0} \tag{3.43}$$

The main difference is that we do not compute the Jacobian with respect to the state $\boldsymbol{x}$ which depends on its representation. Instead we compute the Jacobian with respect to the correction step $\boldsymbol{\epsilon}$ on the manifold, i.e. the tangent space at $\boldsymbol{x}$.

The slightly modified optimization method is described in algorithm 3.

Algorithm 3 Levenberg-Marquardt optimization ”on the manifold”

```
1: input: measurements $\boldsymbol{y}$
2: $\boldsymbol{x}$ := initial estimate
3: while not converged do
4:     compute Jacobian $\boldsymbol{J}(\boldsymbol{x}) = \left. \frac{\partial \boldsymbol{r}(\boldsymbol{x} \boxplus \boldsymbol{\epsilon})}{\partial \boldsymbol{\epsilon}} \right|_{\boldsymbol{\epsilon}=0}$ evaluated at current estimate $\boldsymbol{x}$
5:     compute step $\boldsymbol{h}$ by solving the normal equations $(\boldsymbol{J}^T \boldsymbol{J} + \lambda \boldsymbol{I})\boldsymbol{h}_{lm} = -\boldsymbol{J}^T \boldsymbol{r}$
6:     $\boldsymbol{x} := \boldsymbol{x} \boxplus \boldsymbol{h}$
7: end while
8: output: estimate $\boldsymbol{x}$
```

3.2 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is the combination of two of the most essential tasks in mobile robotics at the same time: Localization, i.e. estimating the pose (position and orientation) of a mobile robot given noisy sensor readings and a prior map of the environment, and Mapping, i.e. using noisy sensor readings to build a map of the environment given known robot poses. The simultaneous combination of both is similar to the chicken-egg problem: Localization requires some knowledge about the environment (i.e. a map), and mapping requires successful localization.

We can only briefly outline the history of he most important and relevant aspects here and refer the interested reader to several good existing overviews in Thrun *et al.* (2005), Thrun and Leonard (2008), Bailey and Durrant-Whyte (2006), and Durrant-Whyte and Bailey (2006).

3.2.1 History

Durrant-Whyte and Bailey (2006) date the beginning of SLAM research to 1986, when the robotics and artificial intelligence communities started looking into probabilistic methods. First significant steps towards the goal of SLAM were achieved in Smith and Cheeseman (1986), which describes a method to estimate the probabilistic relationship between any two coordinate frames, given several chains of uncertain relative transformations affecting both, and Leonard and Durrant-Whyte (1991), which identifies and describes the problem of not only representing but also reducing uncertainty for simultaneous localization and mapping using sonar range sensors. The name and acronym SLAM, however, was only coined in the survey Durrant-Whyte *et al.* (1996).

We now briefly describe the two competing solutions to the SLAM problem: SLAM using probabilistic filtering and graph-based SLAM.

3.2.2 SLAM using Probabilistic Filters

Probabilistic filters and predominantly Extended Kalman Filters (EKF, see Thrun *et al.* (2005) for an accesible description) have been ubiquitous in robotics applications for state estimation in general and specifically mobile robot localization from early on. It was therefore only straightforward to try and pose the SLAM problem as a recursive state estimation problem: If the map is modeled as a number of landmarks and their position estimates, we can augment the state to include both the robot pose as in localization, followed by all landmark positions. This is described in full detail in Smith *et al.* (1990).

The major problem of this approach is the fact that the computational complexity of EKF is quadratic in the dimension of the state and thus the number of landmarks. One way of alleviating this limitation is based on a *factored representation* that decomposes the problem of estimating robot pose and landmark positions into estimating the posterior over the robot path and estimating the locations of landmarks conditioned on the path estimate, thich is called *FastSLAM* and described in Montemerlo *et al.* (2002). FastSLAM estimates the path posterior using a modified particle filter, where each particle in turn contains one Kalman filter per landmark to estimate its location. In conjunction with a tree-based representation lookup structure for all Kalman filters of each particle, the authors claim a computational complexity logarithmic in the number of landmarks.

3.2.3 Graph-Based SLAM

An alternative to representing the SLAM problem as a dynamic Bayesian network is a graph-based representation: Robot poses $\boldsymbol{x}_t$ at certain times t and landmark positions are nodes in the SLAM graph, landmark measurements or relative pose measurements are edges connecting and constraining the corresponding nodes. Solving the SLAM problem then corresponds to finding the best parameters for the nodes such that they are maximally consistent with their edges (i.e. the actual measurements are close to the expected measurements). If measurements can be considered functions of parameters of their related nodes with added Gaussian or similar (see sect. 3.1.4) noise, this optimization problem can be formulated as a least squares problem and optimized using the techniques described in sect. 3.1. This idea was first mentioned in a robotics-related publication[1] in Lu and Milios (1997), where the authors describe how to build and optimize a network of relative pose constraints derived from pairwise registration of 2D laser scans. It took some time for Graph-based SLAM to become accepted, but it is today considered the state-of-the-art method of choice in many applications. A good introduction to graph-based SLAM can be found in Grisetti *et al.* (2010b).

[1]The idea of optimizing a graph built from measurements was not new at all: *Bundle Adjustment*, i.e. the optimization of a graph consisting of camera poses and landmark positions, was well-known in computer vision years before already, see e.g. Granshaw (1980).

Chapter 4

Monocular Visual Simultaneous Localization and Mapping

4.1 Problem Overview

The term *monocular* visual Simultaneous Localization and Mapping (visual SLAM) describes SLAM using only one single camera (greek monos: one, lat. oculus: eye, monocular: ”with one eye”). Relying on a single camera alone is especially intriguing for small flying robots because of the simple sensor setup that allows to build very light-weight systems. The downside of relying on a monocular camera alone, however, is the systematic limitation of not being able to observe metric scale: Since a monocular system observes its environment only through images obtained by perspective projection, it cannot tell the difference between an object of size s at distance d or the same object of size $t \cdot s$ at distance $t \cdot d$. This problem of scale ambiguity and some methods to cope with it are described in more detail in section 4.6.

4.2 Related Fields

Monocular visual SLAM is closely related to the subfield of multiple view geometry in computer vision, whose history in turn can be traced back to the early beginnings of photogrammetry, which is mainly concerned with “obtaining reliable information about physical objects and the environment through [. . .] images [. . .] and other phenomena” (see Slama *et al.* (1980)). Research areas of photogrammetry include, among others, rectification, stereoscopy and aerotriangulation. Methods developed in photogrammetry are often tools to be used by a trained photogrammetrist and do not have to operate fully autonomously without any interaction.

Computer vision, on the other hand, is concerned with automatically extracting information from digital images. The subfield of multiple view geometry studies methods to infer the geometry of a scene that was captured in a number of digital images. It is by now a rather mature field with several comprehensive textbooks available to the interested reader (e.g. Faugeras *et al.* (2001), Hartley and Zisserman (2004), Szeliski (2010)).

Notable examples are systems that can automatically build accurate and visually pleasing 3D reconstructions from a number of digital photographs, e.g. photo tourism (Snavely *et al.* (2006)). We will later see that many methods, especially bundle adjustment, are also applicable to SLAM (see section 4.4.4).

There is one main difference between photogrammetry or multiple view geometry on the one hand and visual SLAM on the other hand, however: SLAM processes sensor streams, i.e. timestamped sequences of images provided at a high frame rate, instead of a sparse number of views captured with a wide baseline. The map of a visual SLAM system is typically similar to the result of applying multiple view geometry algorithms: It almost always consists of a set of keyframes and map points that were seen in multiple keyframes. But a SLAM system is expected to produce pose estimates for each image in the sensor stream.

There is also the term structure from motion (SfM) in computer vision, which is also concerned with reconstructing a 3D model from a stream of images (mapping), for which camera poses (localization) are an implicit byproduct. Structure from motion algorithms, however, are “fundamentally offline in nature, analyzing a complete image sequence” Davison (2003), whereas SLAM is typically required to run and produce its results online and in real-time.

4.3 Related Work

Almost all approaches to solving the monocular visual SLAM problem can be divided into two categories: They are either based on probabilistic filtering or incremental optimization. The former was originally more popular among robotics-related groups since it allows the direct application of methods well-known in robotics, e.g. Kalman filtering, whereas the latter stems from multiple view geometry and was thus preferred among people with a background in computer vision. Differences and similarities as well as their respective advantages and drawbacks are described in great detail in Strasdat *et al.* (2010).

We will now only briefly describe the most prominent example of filtering. A more detailed description of the most prominent example of optimization for visual SLAM follows in section 4.4, since this is the path chosen for the rest of this work.

MonoSLAM

MonoSLAM Davison *et al.* (2007) is an open-source[1] monocular visual SLAM system. It is best described as the application of EKF-SLAM (see e.g. Thrun *et al.* (2005)) to a system with a monocular camera that can observe visual landmarks. Its original version was described in Davison (2003) with subsequent improvements using wide-

[1]The source code is published at `http://www.doc.ic.ac.uk/~ajd/Scene/`

angle lenses (Davison *et al.* (2004)) and representing landmarks using locally planar 3D patches instead of 2D image templates (Molton *et al.* (2004)).

MonoSLAM allowed performing monocular visual SLAM in real-time on a desktop computer in 2003 already, a great achievement for the time. It has some severe limitations, however: Due to the computational complexity of EKF-SLAM, it can only consider a very limited number of visual landmarks at the same time. The original author in Davison *et al.* (2007) proposes "a number in the region of 12" in order not to overburden the processor. Visual landmarks thus regularly have to be dropped from the EKF state, at which point they are considered neither for localization nor for mapping any further.

4.4 Parallel Tracking and Mapping

Parallel Tracking and Mapping (PTAM) as described in Klein and Murray (2007) is a keyframe- and optimization-based monocular visual SLAM system implemented Georg Klein. It was originally intended for augmented reality applications in small environments (e.g. an office desk), which could serve as a virtual playing field.

Since PTAM allows accurate real-time localization at camera rate (typically 30Hz) even when running on constrained computers, it quickly drew the attention of robotics researchers, especially those working on micro aerial vehicles.

4.4.1 Architecture

PTAM uses a key-frame based map, i.e. it models the world as a collection of keyframes (images plus derived data, e.g. the pose at which it was recorded) and map points (distinctive image points that were measured in multiple keyframes and triangulated). The main purpose of the map in PTAM is to allow camera localization: It is concerned neither with occupancy nor traversability.

PTAM uses two threads for two separate tasks: The tracking thread is responsible for estimating the camera pose (i.e. localization) at a high rate with respect to the map. At the same time, the mapping thread builds a keyframe-based map and optimizes keyframe poses and map point positions to minimize the overall reprojection error.

This separation in two threads, combined with very efficient implementations of many state-of-the-art methods, made PTAM a remarkable piece of software that is widely used in robotics applications.

4.4.2 Tracking

The tracking thread iteratively tries to estimate the current camera pose by finding map points in each captured image. For efficiency reasons, it internally estimates the *inverse* camera pose, i.e. the transform ${}^{C_t}\boldsymbol{T}_W$ describing the relative pose of a fixed world-frame W with respect to the camera frame C_t at time t. Tracking involves the following steps:

Prediction Based on Motion Model

PTAM uses a constant decaying velocity motion model for predicting the movement of the camera. This means that for the *a priori* velocity $\overline{\boldsymbol{v}}_t$ it assumes the camera to move at the same velocity $\boldsymbol{v}_{t-1}$ as before , except for a decaying factor $\alpha < 1$ slowing it down.

$$\overline{\boldsymbol{v}}_t = \alpha \cdot \boldsymbol{v}_{t-1} = \alpha \cdot \log \left({}^{C_{t-2}}\boldsymbol{T}_W \cdot {}^{C_{t-1}}\boldsymbol{T}_W^{-1} \right) / \Delta t \tag{4.1}$$

The *a priori* pose estimate ${}^{C}\overline{\boldsymbol{T}}_{W_t}$ is then computed as:

$${}^{C_t}\overline{\boldsymbol{T}}_W = \exp\left(\overline{\boldsymbol{v}}_t \cdot \Delta t\right) \cdot {}^{C_{t-1}}\boldsymbol{T}_W \tag{4.2}$$

This is a suitable motion model for smooth motions, which do not contain abrupt changes.

Efficient Second-Order Minimization for Fast Rotations

Using the basic motion model described above, PTAM computes a reasonable a priori pose estimate in most cases. Abrupt movements, however, are not covered by the model. Fast rotations, especially, cause large changes in the 2D image and thus pose a difficult challenge for tracking features.

This is tackled in PTAM by applying efficient second-order minimization (ESM, see Benhimane and Malis (2004)) to downsampled versions of successive camera images in order to estimate the 2D transformation $\boldsymbol{T}_{ESM} \in \mathrm{SE}(2)$ between both which minimizes the zero mean sum of squared differences (ZMSSD) error. It then tries to find the 3D camera rotation $\boldsymbol{R}_{ESM} \in \mathrm{SE}(3)$ that explains the image motion $\boldsymbol{T}_{ESM}$ best. If ESM-tracking is enabled, it overwrites the predicted rotation according to $\boldsymbol{R}_{ESM}$.

Finding Map Points

The main step in tracking is finding map points within the current image. PTAM first finds interest points, in this case FAST-10 corners as defined in Rosten and Drummond (2006), on multiple levels of the image pyramid (see Tanimoto and Pavlidis (1975)) of the current image. Out of all map points visible according to the a priori pose estimate and camera calibration, it selects a limited number of candidates and warps their source patch by an affine transformation to account for a change in viewpoint. Finally, it performs guided matching of map points to locations of interest points: For each map point candidate, it considers interest points in a circular search window around its expected image position and compares the ZMSSD between warped source patch and image patches centered at the interest points. If the lowest ZMSSD is below a threshold value, it is considered a match and the image location is further refined by the inverse compositional approach from Baker and Matthews (2001).

Nonlinear Optimization

Given several map points that were successfully found in the current camera image at pixel location $\boldsymbol{u}_i$ and their corresponding 3D position in map coordinates ${}^{W}\boldsymbol{p}_i$, PTAM estimates the optimal inverse camera pose ${}^{C_t}\boldsymbol{T}_W$ relative to the map such that it minimizes the total 2D reprojection errors in a robust least squares sense:

$$ {}^{C_t}\boldsymbol{T}_W{}' = \arg\min_{{}^{C_t}\boldsymbol{T}_W} \sum_{i\in S} \rho\left(\frac{\boldsymbol{e}_i}{\sigma_i}\right) \tag{4.3}$$

Where ρ is a robust loss function as described in section 3.1.4 and $\boldsymbol{e}_i$ is the 2D projection error in of measurement i in pixel coordinates, using the notation introduced in section 2.1:

$$\boldsymbol{e}_i = \boldsymbol{u}_i - h\left({}^{C_t}\boldsymbol{T}_W \cdot {}^{W}\boldsymbol{p}_i\right) \tag{4.4}$$

If we consider a correction $\boldsymbol{\mu} \in \mathfrak{se}(3)$ applied to the inverse camera pose along its tangent space, this turns into:

$$\begin{aligned} \boldsymbol{e}_i(\boldsymbol{\mu}) &= \boldsymbol{u}_i - h\left(\left({}^{C_t}\boldsymbol{T}_W \boxplus \boldsymbol{\mu}\right) \cdot {}^{W}\boldsymbol{p}_i\right) && (4.5)\\ &= \boldsymbol{u}_i - h\left(\exp(\boldsymbol{\mu}) \cdot {}^{C_t}\boldsymbol{T}_W \cdot {}^{W}\boldsymbol{p}_i\right) && (4.6)\end{aligned}$$

Since this optimization is performed by robust nonlinear least-squares optimization *on the manifold* $\mathrm{SE}(3)$ (see section 3.1.6), the reprojection error considering actual iterative minimization procedure is computed as:

$$\begin{aligned} \boldsymbol{\mu}' &\leftarrow \arg\min_{\boldsymbol{\mu}} \sum_{i\in S} \rho\left(\frac{\boldsymbol{e}_i(\boldsymbol{\mu})}{\sigma_j}\right) && (4.7)\\ {}^{C_t}\boldsymbol{T}_W &\leftarrow \exp(\boldsymbol{\mu}') \cdot {}^{C_t}\boldsymbol{T}_W = {}^{C_t}\boldsymbol{T}_W \boxplus \boldsymbol{\mu}' && (4.8)\end{aligned}$$

Iterative application of the equations above corresponds to algorithm 3: Equation (4.7) is computed by solving the normal equations, which requires the Jacobian of $\boldsymbol{e}_i$ which is derived later in section 4.4.4. PTAM uses the Tukey biweight function as the robust objective function Obj by default.

Map Initialization

In original PTAM, the user has to intervene to interactively initialize the map by pointing the camera at a first frame, hitting a key, slowly moving the camera sideways, and hitting another key. This provides PTAM with the two first keyframes. Correspondences are determined by tracking some corners from frame to frame using sparse optical flow (hence requiring slow motions) and used to triangulate a set of initial map points. Once this initialization is done, regular tracking as described above begins. Since the distance of the translation between the first two keyframes is unknown, it is assumed to be 1. Monocular SLAM alone cannot recover the correct metric scale, so reconstructed map points and pose estimates are scaled by an unknown scale factor that depends on the translation between the first pair of keyframes.

4.4.3 Mapping

The mapping thread is responsible for adding new keyframes, deciding on and triangulating new map points and, most importantly, local and global optimization of the map, i.e. the SLAM graph.

Deciding About New Keyframes

A new keyframe is added whenever tracking succeeded and the camera is more than a certain threshold way from the closest keyframe. This threshold is chosen as a ratio of the mean scene depth to enforce a minimum baseline between keyframes, which is required for successful triangulation.

Adding New Keyframes

If a new keyframe should be added, PTAM will first try to find some additional map points, since the tracking thread limits the maximum number of candidates considered due to its strict real-time constraint.

After that it tries to create new map points from interest points in the new keyframe by triangulation with the one old keyframe closest to the new one. Since the depth of new map points is initially unknown, it tries to find these within the old keyframe by checking interest points close to their corresponding epipolar line within the one old keyframe. Comparison is performed using ZMSSD without affine warping. The search length along the epipolar line can be limited by only considering possible depth values that are likely to be found given the distribution of depths of existing map points found in the new keyframe.

If a good enough match is found, an initial 3D position estimate of the map point is computed by initialization.

Local and Global Map Optimization

Keyframe pose estimates obtained from tracking and map point position estimates obtained by triangulation are quickly computed but in general not optimal in accuracy, since they only rely on a limited amount of information: Ideally, we should optimize the full SLAM graph for optimal keyframe pose and map point position estimates.

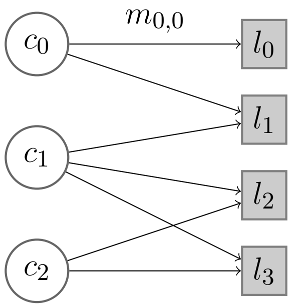


Figure 4.1: Example of a bipartite Bundle Adjustment graph

In monocular visual SLAM, optimizing the full SLAM graph is also called bundle adjustment, which is so important that we describe it in more detail in sect. 4.4.4. PTAM applies bundle adjustment in a first local step optimizing a local window (i.e. submap) consisting of the latest keyframe and its n closest neighbors, and a second global step optimizing the full SLAM graph. The idea behind using two steps is that global optimization is computationally expensive but substantial changes are expected mainly close to the most recent keyframes. Local optimization can thus quickly improve inaccuracies within the latest parts of the map while global optimization spends much computational power for the expensive steps towards optimal accuracy.

If keyframes are added at a rate that does not allow global adjustment between successive keyframes to finish, at least local bundle adjustment is likely to succeed and already significantly improve map accuracy already.

Map Maintenance

Whenever the mapping thread is done with both local and global bundle adjustment, it will spend some time on map maintenance, i.e. trying to identify and remove outlier measurements and finding map points in more old keyframes.

4.4.4 Bundle Adjustment

Bundle Adjustment (BA) is the problem of finding optimal estimates of both camera poses and landmark positions given several measurements of landmarks in camera images. It is a well-studied problem with a long history in computer vision and photogrammetry. A good survey of the most relevant literature can be found in Triggs *et al.* (2000). The problem can be represented as a bipartite SLAM graph with camera pose nodes on one side, landmark nodes on the opposite side, and measurement edges connecting related cameras and landmarks: A camera pose node c_i and landmark node l_j are connected by a measurement $m_{i,j}$ if and only if landmark l_j was seen by camera c_i. A toy example of such a graph is illustrated in Fig. 4.1

Optimal estimates of camera poses $\boldsymbol{a}$ and landmark positions $\boldsymbol{b}$ should generally minimize a cost function

$$\sum_{i=1}^{N} \sum_{j \in S_i} \rho \left(\frac{\boldsymbol{e}_{i,j}(\boldsymbol{a}_i, \boldsymbol{b}_i)}{\sigma_{i,j}} \right) \tag{4.9}$$

Where $\boldsymbol{e}_{i,j} = \boldsymbol{m}_{i,j} - \widehat{\boldsymbol{m}_{i,j}}$ is the reprojection error of measurement $\boldsymbol{m}_{i,j}$, $\widehat{\boldsymbol{m}_{i,j}}(\boldsymbol{a}_i, \boldsymbol{b}_j)$ is the expected measurement given the current estimates of camera pose $\boldsymbol{a}_i$ and landmark position $\boldsymbol{b}_i$, $\sigma_{i,j}$ is the uncertainty (i.e. assumed standard deviation) of the measurement according to our probabilistic measurement model and ρ is a robust loss function.

This is a robust non-linear least squares optimization problem and can be solved using the methods described in sect 3.1.4. In PTAM, this optimization is performed using a variant of the Levenberg-Marquardt algorithm that applies a slightly different augmentation using the damping parameter also mentioned in Hartley and Zisserman (2004), which replaces the matrix $\left(\boldsymbol{J}^T\boldsymbol{J} + \mu \boldsymbol{I}\right)$ in the normal equations by the matrix $\boldsymbol{J}^T\boldsymbol{J}$ with its diagonal multiplied by $(1+\mu)$.

$$\left(\boldsymbol{J}^T\boldsymbol{J} + \mu \cdot \mathrm{diag}(\boldsymbol{J}^T\boldsymbol{J})\right) \boldsymbol{h}_{lm} = -\boldsymbol{J}^T\boldsymbol{f} \tag{4.10}$$

Parameter Partition

The normal equations are a linear system of size $n \times n$ with n being the total parameter size or $n = 6n_c + 3n_p x$ with n_c cameras and n_p points. Solving this system naively quickly becomes infeasible for real-time application. Major speedups can be achieved by considering the sparsity of the Jacobian and pseudo-Hessian and the fact that they are block matrices with a specific structure. This structure is illustrated for the example graph of Fig. 4.1 in Fig. 4.2: Each measurement adds two rows to J, which are only non-zero in columns corresponding to the camera pose at which it was measured and the position of the related landmark. It can also be partitioned into two parts:

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \end{pmatrix} \tag{4.11}$$

Where $\boldsymbol{A}$ is the Jacobian of the error vector $\boldsymbol{f}$ with respect to camera poses and $\boldsymbol{B}$ with respect to landmark positions. It follows that the matrix $\boldsymbol{J}^T\boldsymbol{J}$ also exhibits a partitioned structure:

$$\boldsymbol{J}^T\boldsymbol{J} = \begin{pmatrix} \boldsymbol{A}^T\boldsymbol{A} & \boldsymbol{A}^T\boldsymbol{B} \\ \boldsymbol{B}^T\boldsymbol{A} & \boldsymbol{B}^T\boldsymbol{B} \end{pmatrix} = \begin{pmatrix} \boldsymbol{U} & \boldsymbol{W} \\ \boldsymbol{W}^T & \boldsymbol{V} \end{pmatrix} \tag{4.12}$$

The normal equations (4.10) can thus also be written as:

$$\begin{pmatrix} \boldsymbol{U}^\star & \boldsymbol{W} \\ \boldsymbol{W}^T & \boldsymbol{V}^\star \end{pmatrix} \begin{pmatrix} \boldsymbol{h}_a \\ \boldsymbol{h}_b \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_a \\ \boldsymbol{f}_b \end{pmatrix} \tag{4.13}$$

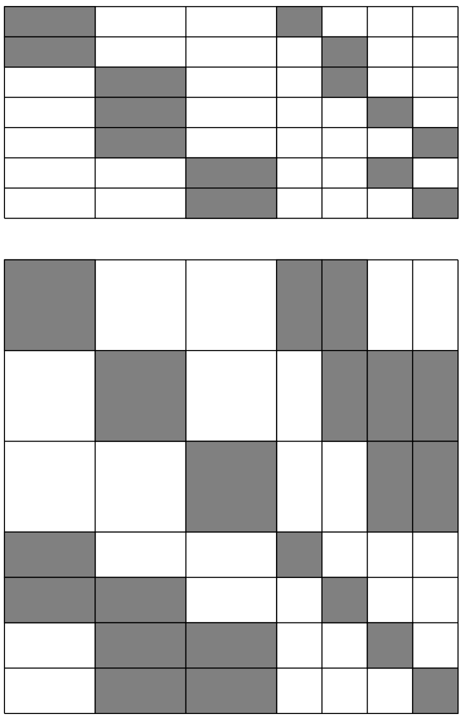


Figure 4.2: Block structure of Jacobian $\boldsymbol{J}$ (top) and Pseudo-Hessian $\boldsymbol{J}^T\boldsymbol{J}$ (bottom) for the example graph shown in figure 4.1.

Where $\boldsymbol{U}^\star$ and $\boldsymbol{V}^\star$ correspond to the matrices $\boldsymbol{U},\boldsymbol{V}$ with their diagonals multiplied by $(1+\mu)$. Multiplying the previous equation by the following matrix

$$\begin{pmatrix} \boldsymbol{I} & -\boldsymbol{W}\left(\boldsymbol{V}^\star\right)^{-1} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \tag{4.14}$$

from the left-hand side yields:

$$\begin{pmatrix} \boldsymbol{U}^\star - \boldsymbol{W}\left(\boldsymbol{V}^\star\right)^{-1}\boldsymbol{W}^T & \boldsymbol{0} \\ \boldsymbol{W}^T & \boldsymbol{V}^\star \end{pmatrix} \begin{pmatrix} \boldsymbol{h}_a \\ \boldsymbol{h}_b \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_a - \boldsymbol{W}\left(\boldsymbol{V}^\star\right)^{-1}\boldsymbol{f}_b \\ \boldsymbol{f}_b \end{pmatrix} \tag{4.15}$$

Solving eq. (4.15) now involves consecutively solving two much smaller systems compared to eq. (4.13), which is in general much faster.

Measurement Jacobians

The actual numbers for each entry in the Jacobian $\boldsymbol{J}$ could be computed automatically using numerical differentiation, which is too slow for real-time optimization, or automatic differentiation Griewank (1989). Since analytical derivatives do exist and can be computed, however, using analytical Jacobians is preferable for both fast convergence and computation. The Jacobians are not derived in Klein and Murray (2007) so we derive these here for future reference.

When considering only one measurement $m_{i,j}$ and its related camera pose c_i and landmark position p_i, its reprojection error in 2D pixel coordinates is:

$$\boldsymbol{e}_{i,j} = \boldsymbol{u}_{i,j} - \widehat{\boldsymbol{u}}_{i,j} = \boldsymbol{u}_{i,j} - h\left({}^{C_i}\boldsymbol{T}_W \cdot {}^{W}\boldsymbol{p}_j\right) \tag{4.16}$$

The relevant non-zero blocks of its Jacobian $\boldsymbol{A}_{i,j}$ with respect to change in camera pose $\boldsymbol{C}_i$ and $\boldsymbol{B}_{i,j}$ with respect to landmark position $\boldsymbol{p}_i$ are:

$$\begin{align}
\boldsymbol{A}_{i,j} &= -\frac{\partial h\left(e^{\epsilon} \oplus \boldsymbol{C}_i \oplus \boldsymbol{p}_j\right))}{\partial \epsilon} \tag{4.17}\\
&= -\left.\frac{\partial h(\boldsymbol{p}')}{\partial \boldsymbol{p}'}\right|_{\boldsymbol{p}'=\boldsymbol{C}_i \oplus \boldsymbol{p}_j} \cdot \frac{\partial e^{\epsilon} \oplus \boldsymbol{C}_i \oplus \boldsymbol{p}_j}{\partial \epsilon} \tag{4.18}\\
&= -\left.\frac{\partial h(\boldsymbol{p}')}{\partial \boldsymbol{p}'}\right|_{\boldsymbol{p}'=\boldsymbol{C}_i \oplus \boldsymbol{p}_j} \cdot \left(\boldsymbol{I}_3 \quad -[\boldsymbol{C}_i \oplus \boldsymbol{p}_j]_{\times}\right) \tag{4.19}
\end{align}$$

$$\begin{align}
\boldsymbol{B}_{i,j} &= -\frac{\partial h\left(\boldsymbol{C}_i \oplus \boldsymbol{p}_j\right))}{\partial \boldsymbol{p}_j} \tag{4.20}\\
&= \left.\frac{\partial h(\boldsymbol{p}')}{\partial \boldsymbol{p}'}\right|_{\boldsymbol{p}'=\boldsymbol{C}_i \oplus \boldsymbol{p}_j} \cdot \frac{\boldsymbol{C}_i \oplus \boldsymbol{p}_j}{\partial \boldsymbol{p}_j} \tag{4.21}\\
&= -\left.\frac{\partial h(\boldsymbol{p}')}{\partial \boldsymbol{p}'}\right|_{\boldsymbol{p}'=\boldsymbol{C}_i \oplus \boldsymbol{p}_j} \cdot \boldsymbol{R}_{C_i} \tag{4.22}
\end{align}$$

We have shown in section 2.1.5 that the camera projection h can be modeled as an idealized perspective projection, distortion, and finally a linear projection (cf. equation (2.11)):

$$h(\boldsymbol{p}) = u\left(d\left(n\left(\boldsymbol{p}\right)\right)\right) \tag{4.23}$$

In order to compute the Jacobian of h missing above, we can thus apply the chain rule and substitute $\boldsymbol{C}_i \oplus \boldsymbol{p}_j = \boldsymbol{q}$:

$$\left.\frac{\partial h(\boldsymbol{p}')}{\partial \boldsymbol{p}'}\right|_{\boldsymbol{p}'=\boldsymbol{q}} = \left.\frac{\partial u(\boldsymbol{d}')}{\partial \boldsymbol{d}'}\right|_{\boldsymbol{d}'=d(n(\boldsymbol{q}))} \cdot \left.\frac{\partial d(\boldsymbol{n}')}{\partial \boldsymbol{n}'}\right|_{\boldsymbol{n}'=n(\boldsymbol{q})} \cdot \left.\frac{\partial n(\boldsymbol{p}')}{\partial \boldsymbol{p}'}\right|_{\boldsymbol{p}'=\boldsymbol{q}} \tag{4.24}$$

With the Jacobian of the idealized perspective projection:

$$\left.\frac{\partial n(\boldsymbol{p}')}{\partial \boldsymbol{p}'}\right|_{\boldsymbol{p}'=\boldsymbol{q}} = \begin{pmatrix} \frac{1}{q_z} & 0 & -\frac{1}{q_z^2} \\ 0 & \frac{1}{q_z} & -\frac{1}{q_z^2} \end{pmatrix} \tag{4.25}$$

And the Jacobian of the linear projection:

$$\left.\frac{\partial u(\boldsymbol{d}')}{\partial \boldsymbol{d}'}\right|_{\boldsymbol{d}'=d(n(\boldsymbol{q}))} = \begin{pmatrix} f_x & \gamma \\ 0 & f_y \end{pmatrix} \tag{4.26}$$

The Jacobian matrix of the distortion depends on the desired distortion model. For the ATAN model, it can easily be computed and is still rather compact:

$$\left.\frac{\partial d(\boldsymbol{n}')}{\partial \boldsymbol{n}'}\right|_{\boldsymbol{n}'=\boldsymbol{n}} = \frac{1}{r_u}\begin{pmatrix} n_x^2 & n_x n_y \\ n_x n_y & n_z^2 \end{pmatrix} \left.\frac{\partial r_d(r_u')}{\partial r_u'}\right|_{r_u'=r_u} + r_d \boldsymbol{I}_2 \tag{4.27}$$

Where $r_u = \sqrt{n_x^2 + n_y^2}$ is the undistorted radius, r_d is the distorted radius, and the remaining derivative above can be computed from the definition of the distorted radius in section 2.1.4:

$$\frac{\partial r_d(r_u)}{\partial r_u} = \frac{2\tan(\frac{\omega}{2})}{\omega\left(4r_u^2\tan^2(\frac{\omega}{2})\right)} \tag{4.28}$$

4.5 Porting and Extending Parallel Tracking and Mapping

Our first steps towards being able to use PTAM on our robots involved porting it to the Robot Operating System[2] (ROS, see Quigley *et al.* (2009)), a robotics framework and middleware currently used in almost all robotics research laboratories worldwide. We implemented a ROS-wrapper around PTAM to allow using camera images provided via ROS messages instead of accessing the camera directly. This allows using PTAM with any camera supported by a driver for ROS, with data recorded on any robot using the rosbag functionality for logging, and even when streaming camera images over the network.

Since software on robots often needs to run in a headless mode without being able to use any GUI, we also implemented an alternative visualization using ROS visualization messages so we can stream and visualize the current state of keyframes and map points together with other properties of a robot on a separate ground station computer.

This enabled autonomous navigation of a micro aerial vehicle with only a monocular camera which led to multiple interesting projects which are only tangentially related to

[2] `http://www.ros.org`

the scope of this dissertation. Yang *et al.* (2012), Yang *et al.* (2013a), Yang *et al.* (2013b), Yang *et al.* (2014a), Yang *et al.* (2014c), Yang *et al.* (2014b)

4.6 System-Inherent Limitations of Monocular SLAM

As mentioned before, a monocular camera alone cannot observe the global metric scale of its motion and its environment: There will always be a scale ambiguity. This ambiguity, however, has to be resolved in order to provide useful information, e.g. pose estimates, to a mobile robot.

In the case of a flying robot, we might safely assume that it has at least some prior information about the location where it takes off, e.g. a helicopter landing pad of known size. As long as the camera sees this known landing pad, the relative pose of the camera including the full metric scale can be inferred. Using a modified version of our object detection method originally intended to detect number signs on wheeled robots described in Scherer *et al.* (2011), we were able to initialize PTAM metrically correct by relying on the relative pose to the landing pad for the two first keyframes. This enabled autonomous takeoff, navigation, and landing of a small MAV using a monocular camera as its main sensor in Yang *et al.* (2012) and Yang *et al.* (2013a).

Another way to cope with this scale ambiguity is mounting a complementary metric sensor an the MAV. A cheap, light-weight and thus popular choice for MAVs is the combination of a downward-looking monocular camera combined with an ultrasonic sensor that can measure the distance to the ground. It has been shown multiple times that this allows to get metric visual odometry measurements, e.g. in Grabe *et al.* (2012). In a similar vain, depth or stereo cameras also provide metric measurements of the scene geometry, which is described in Scherer *et al.* (2012) or in chapter 5 in more detail.

Finally, an inertial measurement unit (IMU) can be used to infer the otherwise ambiguous scale factor. Weiss *et al.* (2012) demonstrate that the scale ambiguity can be resolved by using pose estimates with unknown scale obtained from a monocular odometry system within an extended Kalman Filter that estimates among others the current scale factor within its state.

Chapter 5

Using Depth in Visual Simultaneous Localization and Mapping

In this chapter, we propose a novel combination of 2D information from color or intensity images with depth measurements as obtained by depth cameras for visual simultaneous localization and mapping. We described and published most aspects of this chapter in a more compact form in Scherer *et al.* (2012) for the first time.

5.1 Motivation

RGBD cameras, which are cheap, lightweight, easy to use with open-source drivers, and provide rich information about the environment at high rates, are justifiably considered outstandingly suitable sensors for solving the simultaneous localization and mapping task on mobile robots.

For us, the major goals for SLAM using RGBD cameras were high efficiency, to allow real-time operation on computationally constrained hardware, accuracy of localization and mapping, and robustness with respect to failures in measuring depth, which is explained in more detail in section 5.3.

5.2 Related Work

Related approaches to SLAM with RGBD cameras in general belong to one of two categories: They can be based on feature matching or dense registration. Some approaches even combine both.

5.2.1 Feature Matching

The first category works by matching sparse image features between RGB (i.e. intensity) images. If there are valid depth measurements at the relevant pixel locations in both depth images, the 3D positions of both image features can be reconstructed, which leads to a set of 3D to 3D point correspondences. The relative pose of two RGBD frames can be computed from at least three such 3D to 3D point correspondences using one of the closed-form solutions described in Eggert *et al.* (1997), ideally within a RANSAC scheme to weed out possible outliers. In computer vision, estimating the relative transformation between 3D to 3D point correspondences is a well-known problem called absolute orientation (see Horn (1987)).

The first such approach to RGBD-SLAM was demonstrated by Henry *et al.* (2010). After successful initial registration as described above, they refine the relative transformation using iterative closest point registration (ICP, see Besl and McKay (1992) Chen and Medioni (1992)) of the full point clouds reconstructed from both depth images. They also employ the same registration technique for registering frame upon detected loops and optimize the map using pose graph optimization. An open-source implementation of this approach is described in Engelhard *et al.* (2011), which the authors call RGBD-SLAM.

5.2.2 Dense Methods

We call the second category of RGBD-SLAM approaches *dense* methods, since they consider the whole RGBD image instead of only a *sparse* set of interest points. If we assume to obtain reliable depth measurements for the full image, we can warp the current RGBD image by applying a rigid body transform in $\mathrm{SE}(3)$ to all of its 3D points until it is as close as possible to a second reference depth image.

In the simplest case, this registration may be performed on point clouds reconstructed from depth images, for which well-known algorithms like iterative closest point (ICP, see Besl and McKay (1992)), its more modern variant generalized ICP (see Segal *et al.* (2009)), and the normal distributions transform (Biber and Straßer (2003)) exist.

When using RGBD images, it is more efficient to minimize error measures defined on the RGBD images directly, since this is a much more compact representation than a colored point cloud. Steinbrucker *et al.* (2011) proposed to maximize photoconsistency, i.e. to minimize the squared intensity error between warped current and reference image for odometry estimation. Whelan *et al.* (2013) showed a real-time implementation of this method on a GPU. A robust variant that can deal with outliers was described in Kerl *et al.* (2013a) and a SLAM system built around this in Kerl *et al.* (2013b).

Compared to sparse methods, dense methods have both advantages and drawbacks: Since they make use of more information contained in the full RGBD image they can be more accurate than sparse methods. This obviously comes at the expense of computation time. Even initialy implementations required a GPU for camera rate tracking (Whelan *et al.* (2013)), current implementations as in Kerl *et al.* (2013a) allow close to camera rate tracking on a powerful CPU at reduced resolutions (QVGA) already. Moore's law predicts that the difference in computation time will likely become negligible in the future. But until then, sparse methods still have their right to exist, especially on computationally constrained systems such as MAVs.

5.3 Main Idea

All of the methods mentioned in section 5.2 require reliable dense depth images measured by the depth camera. Early on in our experiments with RGBD cameras on mobile robots, however, we noticed image pairs similar to the one shown in figure 5.1: Large parts of the environment are missing from the depth image (shown in black) due to depths above the maximum measurement range, challenging material (reflective floor, glass cabinets), and even sunlight. Not considering large parts of such an RGBD image pair at all because of missing depth values would throw away useful information, since the RGB channel (i.e. texture) alone still tells *something* about the environment: At least as much as we could infer using monocular visual SLAM alone.

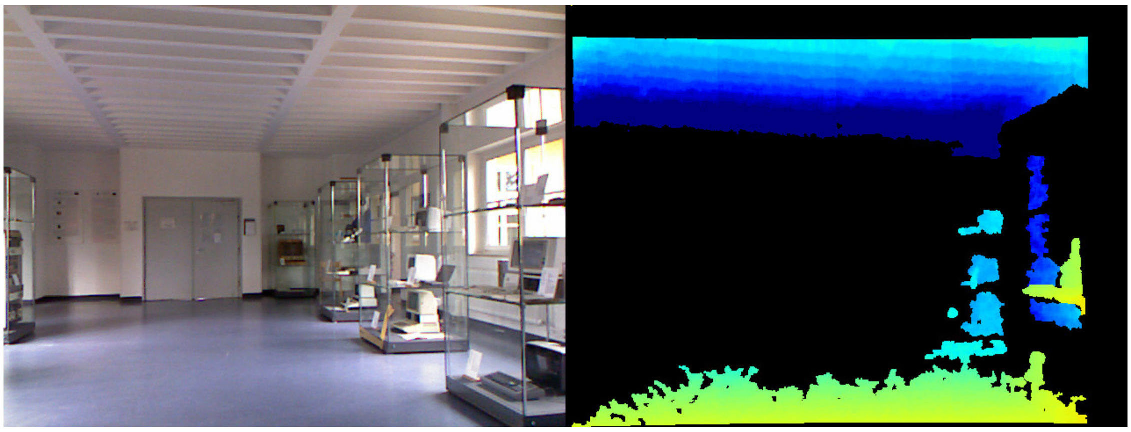


Figure 5.1: RGBD image pair taken on the corridor in our department.

So why not base our work on mononcular visual SLAM, which can be implemented efficiently as seen in chapter 4, and improve it using depth measurements whenever available? We would expect this to be at least as accurate and reliable as using monocular visual SLAM alone. Adding a number of depth measurements should improve its accuracy considerably and solve its systemic limitations described in section 4.6.

5.4 Integrating Depth Information

All measurements as used for both tracking and mapping in original PTAM are always the 2D pixel locations at which map points were found. For combining both 2D and depth information, we considered two options: Minimizing 3D reprojection errors or 2D plus depth reprojection errors.

5.4.1 3D Reprojection Errors

If there is valid depth information available for a measured 2D position, we can reconstruct the full 3D point $\boldsymbol{p}_i$ and compute the error between expected and measured position of a map point in 3D camera coordinates:

$$\boldsymbol{e}_i = \boldsymbol{p}_i - {}^{C_t}\boldsymbol{T}_W \cdot {}^{W}\boldsymbol{p}_i \tag{5.1}$$

Compared to the 2D reprojection error (equation (4.4)), this is much easier, since the error term in this case does not depend on the camera projection model but only on the camera pose. This is the reprojection error minimized by Henry *et al.* (2010) and Engelhard *et al.* (2011) and in iterative closest point (ICP) registration of two point clouds in general. Several closed-form solutions for the regular least-squares optimum can be found in Eggert *et al.* (1997).

Error Jacobians

If we were to use 3D reprojection errors in efficient non-linear least squares optimization of the camera pose ${}^{C_t}\boldsymbol{T}_W$, we would need the non-zero blocks of its Jacobian $\boldsymbol{A}_{i,j}$ with respect to change in camera pose $\boldsymbol{C}_i$ along the manifold, which can be computed as:

$$\begin{aligned}\boldsymbol{A}_i &= -\frac{\partial \left(e^{\epsilon} \cdot {}^{C_t}\boldsymbol{T}_W \cdot {}^{W}\boldsymbol{p}_i\right)}{\partial \epsilon} && (5.2)\\ &= -\left(\boldsymbol{I}_3 \quad - \left[{}^{C_t}\boldsymbol{T}_W \cdot {}^{W}\boldsymbol{p}_i\right]_\times\right) && (5.3)\end{aligned}$$

For bundle adjustment, we also need the Jacobian $\boldsymbol{B}_{i,j}$ of the 3D reprojection error with respect to the landmark position $\boldsymbol{p}_i$, which is:

$$\begin{aligned}\boldsymbol{B}_i &= -\frac{\partial \left({}^{C_t}\boldsymbol{T}_W \cdot {}^{W}\boldsymbol{p}_i\right)}{\partial {}^{W}\boldsymbol{p}_i} && (5.4)\\ &= -\left({}^{C_t}\boldsymbol{R}_W\right) && (5.5)\end{aligned}$$

Note that $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ in this case correspond to equations (4.19) and (4.19) without the factor due to the full camera projection.

Problems

There are two problems with this approach, however: First, we obviously cannot use the 3D reprojection error above for measurements without valid depth information. For these measurements, we are still stuck with 2D reprojection errors, i.e. we would need to mix 2D and 3D errors. Second, we need to estimate the measurement uncertainty of all reprojection errors in order to combine these using heteroskedastic least squares as described in section 3.1.3. Doing this accurately for 3D errors is difficult, since 3D positions are not measured directly but can only be inferred by transforming two other measurements (2D and depth) by applying a nonlinear function. The distribution of 3D errors is thus even more likely to violate the Gaussian noise assumption and any estimate of the covariance matrix of 3D positions, e.g. obtained by propagating measurement covariances through a linearized model, is bound to be inaccurate. This leads us towards the following second option for integrating depth measurements.

5.4.2 Combining 2D and Depth Reprojection Errors

We have already established that we need to keep using 2D reprojection errors when there is no depth available. It is only consistent to just always use 2D reprojection errors and add depth reprojection errors whenever depth is available. This depth reprojection error is simply:

$$\boldsymbol{e}_i = d_i - \left({}^{C_t}\boldsymbol{T}_W \cdot {}^{W}\boldsymbol{p}_i\right)_z \tag{5.6}$$

Where $(\boldsymbol{p})_z$ denominates the z coordinate of point $\boldsymbol{p}$. Depth, after all, is defined as the z coordinate of a point in 3D.

Error Jacobians

We again need the Jacobian matrix of the depth reprojection error $\boldsymbol{e}_i$ with respect to small change $e^{\boldsymbol{\epsilon}}$ in camera pose ${}^{C_t}\boldsymbol{T}_W$ along its tangential manifold:

$$\begin{aligned}
\boldsymbol{A}_i &= -\frac{\partial\left(e^{\epsilon} \cdot {}^{C_t}\boldsymbol{T}_W \cdot {}^{W}\boldsymbol{p}_i\right)_z}{\partial \epsilon} && (5.7)\\
&= -\left(\boldsymbol{I}_3 \quad -\left[{}^{C_t}\boldsymbol{T}_W \cdot {}^{W}\boldsymbol{p}_i\right]_\times\right)_z && (5.8)\\
&= -\begin{pmatrix}0 & 0 & 1 & ({}^{C_t}\boldsymbol{p}_i)_y & -({}^{C_t}\boldsymbol{p}_i)_x & 0\end{pmatrix} && (5.9)
\end{aligned}$$

Which is identical to the last row of equation (5.3). The Jacobian of depth error with respect to its map point's position, similarly, is identical to the last row of equation (5.5):

$$\begin{aligned}
\boldsymbol{B}_i &= -\frac{\partial\left({}^{C_t}\boldsymbol{T}_W \cdot {}^{W}\boldsymbol{p}_i\right)_z}{\partial {}^{W}\boldsymbol{p}_i} && (5.10)\\
&= -\left({}^{C_t}\boldsymbol{R}_W\right)_z && (5.11)
\end{aligned}$$

Error Uncertainty and Normalization

2D and depth reprojection errors are in completely unrelated dimensions: 2D positions are measured in pixels, depth in meters. They can still be combined using heteroskedastic least squares if an estimate of their measurement uncertainty is known. Section 5.6 describes our probabilistic measurement models and how we determined the model parameters.

5.5 Parallel Tracking and Mapping using Depth Measurements

We extended our fork of PTAM running on ROS first mentioned in section 4.5 to enable use of RGBD image pairs. It relies mainly on the RGB part (i.e. the intensity image) and queries the depth image for depth values at the location of interest points (FAST corners) within the intensity image. We extract and store depth values for all FAST corners, which only introduces a negligible overhead in computational load and memory consumption. After that, the depth image may be discarded. We usually still keep it in memory for image pairs that are promoted to keyframes and added to the map for further processing by other tasks.

5.5.1 Map Initialization

With RGB and depth information available, the map can be initialized from a single RGBD frame: Out of all interest points after non-maximum suppression with valid depth values in the initial RGBD frame, we only consider these with a Shi-Tomasi score above a fixed threshold. The best (according to their Shi-Tomasi score) n per pyramid level are promoted to map points if there was no map point at almost the same location on a higher level already.

Since we have depth measurements for all such map points, we can compute their 3D position estimates from the first RGBD frame alone using equation (2.3) and start tracking this map.

5.5.2 Pose Tracking

Pose tracking does not explicitly require depth measurements for metrically correct pose estimates as long as map points are correctly triangulated. Three 3D to 2D correspondences are enough in the minimal case, after all, to recover the camera pose as described in Fischler and Bolles (1981) or DeMenthon and Davis (1992). For ideal accuracy, however, we want to use all information available: Both 2D and depth measurements.

We thus optimize an augmented version of the objective function in equation (4.3) which combines both 2D and depth measurements in the set S:

$$\boldsymbol{\mu}' = \arg\min_{\boldsymbol{\mu}} \sum_{i \in S} \rho\left(\frac{\boldsymbol{e}_i}{\sigma_i}\right) \tag{5.12}$$

Here, $\boldsymbol{e}_i$ may either be a 2D reprojection error in pixels $\boldsymbol{e}_i \in \mathbf{R}^2$ or depth $\boldsymbol{e}_i \in \mathbf{R}$ in meters. The uncertainty estimate σ_i is then chosen according to the model described in section 5.6.

5.5.3 Triangulation when adding Keyfames

New map points are added whenever a keyframe is created. Since PTAM represents map points using their position with respect to a fixed global 3D coordinate frame, they need to be properly triangulated before they can be added to the map. [1]

The original version of PTAM creates new map points from a new keyframe in the following way: Interest points are computed for tracking already, but some were associated with previously existing map points. Thus only interest points that are not too close to the projections of existing map points are considered as candidates for new map points. PTAM then tries to find these map point candidates within the closest previous keyframe by guided matching along its corresponding epipolar line and computes a first estimate of its 3D position by stereo triangulation. In consequence, a map point can only be triangulated and therefore used for tracking after it was measured in at least two keyframes.

When using depth images, we choose new map points in the same way, but we prefer to rely on depth measurements instead of triangulation: If there is a valid depth measurement for a map point candidate, we do not try to find it in a previous keyframe but compute its 3D position estimate using the measured depth value. If there is no valid depth measurement, we fall back to triangulation instead.

5.6 Measurement Uncertainty Models

Heteroskedastic least squares (see section 3.1.3) allows us to combine measurements from various sensors with different measurement uncertainties. In addition to the values actually measured, it also requires an uncertainty estimate for each measurement. Finding an appropriate model for these uncertainty estimates is thus an important issue.

5.6.1 2D Uncertainty

2D pixel position measurements in PTAM are obtained by finding a map point on a certain image pyramid level i of an image. The pixel location within the pyramid level is then represented with respect to the original image to allow using the same camera cal-

[1] Using a different representation, e.g. a camera-fixed representation with inverse depth coordinates as proposed in Montiel *et al.* (2006), it would be possible to initialize a map point from a single 2D measurement as a ray with infinite depth uncertainty.

ibration for all measurements.[2] We can expect the 2D pixel accuracy to be proportional to the size of a pixel on pyramid level i with respect to pixels of the original image s_i

$$\sigma_i = k \cdot s_i \tag{5.13}$$

with a common proportionality constant k. This is implemented in the original version of PTAM already. Since it originally only uses 2D measurements, the actual value of k is irrelevant,[3] however, and thus set to $k = 1$.

When combining 2D and different measurements with uncertainty estimates, the actual value of k does matter, since the other measurements might not depend on k, and choosing the value of k arbitrarily changes the influence of 2D measurements over the others.

We determined a rough estimate for k experimentally in Scherer *et al.* (2012) by running PTAM using only 2D measurements and determining the standard deviation of final reprojection errors after optimization for each pyramid level individually. The constant k can then be determined from each pyramid level i via:

$$k = \frac{\sigma_i}{s_i} \tag{5.14}$$

Averaging over all pyramid levels lead us to an estimate of $k = 0.987$.

5.6.2 Depth Uncertainty

Theoretical Model

Depth measurements in Primesense-based RGBD cameras are obtained internally by measuring the displacement of an infrared dot pattern projected into the scene (see section 2.2.2). The underlying geometry is identical to stereo vision, where disparity p and depth d are known to be connected by focal length f and base line B (see e.g. Szeliski (2010), p. 539):

$$p = \frac{fB}{d} \Rightarrow d = \frac{fB}{p} \tag{5.15}$$

It is safe to assume that the disparity p can be measured with a fixed uncertainty σ_p, which is limited by the template matching accuracy. We can now use the rules of error propagation to compute an approximation of the uncertainty in depth σ_d. In the 1-dimensional

[2]Measurements of 2D pixel positions could instead also considered with respect to their actual pyramid level. This would require using different intrinsic camera calibration parameters (principal point and focal length) for each pyramid level, which is rather inconvenient.

[3]Changing the value of k is identical to modifying the weight of all 2D measurements by the same global factor, which does not affect the outcome of least squares optimization.

case, this is simply (see Taylor (1997), p. 65):

$$\partial q = \left| \frac{dq}{dx} \right| \partial x \tag{5.16}$$

This leads us to:

$$\sigma_d = \left| \frac{\partial d(p)}{\partial p} \right| \cdot \sigma_p \tag{5.17}$$

$$= \left| -\frac{fB}{p^2} \right| \cdot \sigma_p = \frac{d^2}{fB} \cdot \sigma_p \tag{5.18}$$

$$\sigma_d(d) = k_d \cdot d^2 \tag{5.19}$$

Where we replaced the unknown disparity p by depth d since this is the value actually provided by the depth camera. This means the uncertainty of depth measurements is expected to grow proportionally to the square of the actual depth value. As shown above, we can combine all calibration parameters f, B, and σ_p of the model in a single constant k_p, which can be determined in a calibration experiment.

Parameter Identification

The unknown parameter k_d above may be different for each depth camera. It has to be identified in a calibration step. We did this with the following experimental setup: We placed the RGBD camera directly facing a single large planar wall at various distances from the wall ranging from $5 \times 10^{-1}\,\mathrm{m}$ to $3.5\,\mathrm{m}$ and captured calibration images. We then used the LO-RANSAC algorithm described in Chum *et al.* (2004) to robustly fit a plane to each recorded depth image. With knowledge of the plane parameters, we could compute both depth and depth error for a large number of points on these calibration planes.

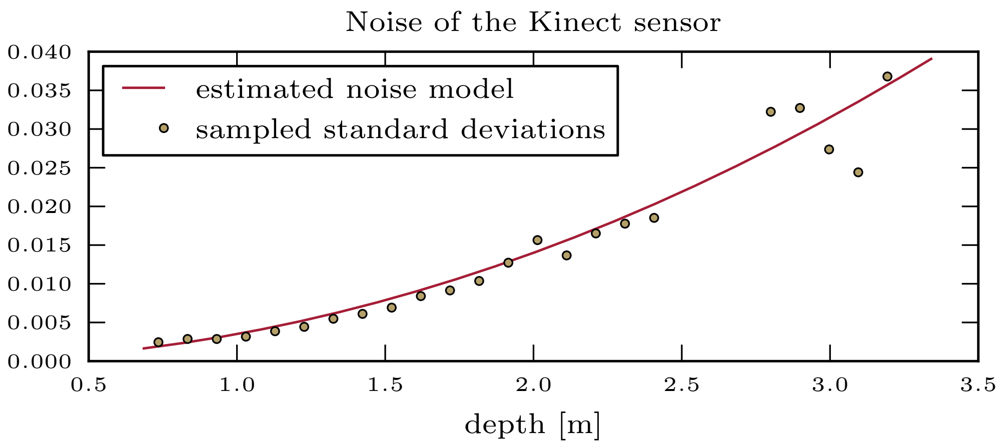


Figure 5.2: Experimentally determined standard deviations of the Microsoft Kinect depth camera over actual depth.

In a second step we sorted all points by their actual depth and collected samples of similar depth in bins of 10cm width. This allowed us to estimate the standard deviation σ_b for each bin, which is illustrated by the dots in figure 5.2. In order to find the parameter k_d, we finally fit the model equation (5.19) to our measurements by minimizing the relative error with respect to the standard deviations estimated from the sampled bins:

$$e_b = \sum_b \frac{\sigma_b - \sigma_d(d_b)}{\sigma_b}$$

Which led us to a value of $k_d = 3.331 \times 10^{-3}\,\mathrm{m}^{-1}$.

Later Experiments

Rauscher *et al.* (2014) evaluated multiple depth and range sensors including the Microsoft Kinect depth camera in more detailed experiments. They chose to fit a slightly different model, though:

$$\sigma_d(d) = a + bd + cd^2$$

Where a and b do not have a theoretical foundation but seem to slightly improve the model accuracy. The corresponding model parameters for the Kinect depth camera are determined to be $a = 7.152 \times 10^{-3}\,\mathrm{m}$, $b = -6.750 \times 10^{-3}$, and $c = 3.296 \times 10^{-3}\,\mathrm{m}^{-1}$. The values of a and b become more and more irrelevant with growing depth values and c is actually very close to the value of k_p obtained as described above, which shows that the two models almost agree.

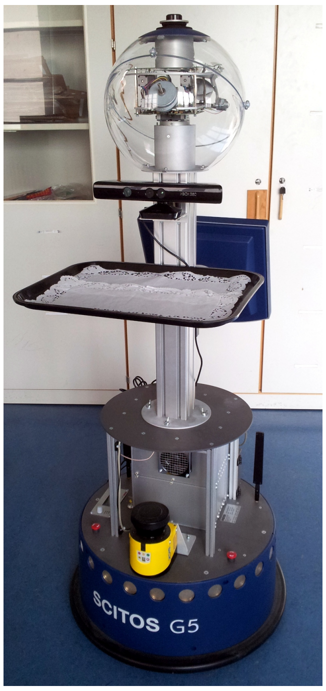


Figure 5.3: Metralabs SCITOS G5 robot with Microsoft Kinect and Sick S300 laser rangefinder used for accuracy evaluation.

5.7 Experiments and Results

Since we initially did not ave access to an external tracking system which could provide us with ground truth camera poses, we mounted a Microsoft Kinect RGBD camera on a Metralabs SCITOS G5 wheeled mobile robot with a differential drive mechanism and a SICK S300 laser rangefinder depicted in figure 5.3. Laser scans and odometry information allowed us to accurately localize the robot within a previously built 2D map of the environment using monte carlo localization (see Fox *et al.* (1999)) and thus provide ground truth pose estimates.

We manually controlled the robot to drive ca. 106 m through a long corridor and a visually challenging museum room with several glass cabinets. Long, weakly textured walls, reflecting surfaces, and a mixture of artificial light and sunlight shining through windows made this a challenging environment. We recorded all data available for further offline processing and evaluation.

Accuracy Evaluation

We use the dataset mentioned above to evaluate and compare localization accuracy using the different improvements introduced by our method.

For the first experiment, we applied the original monocular version of PTAM, but we

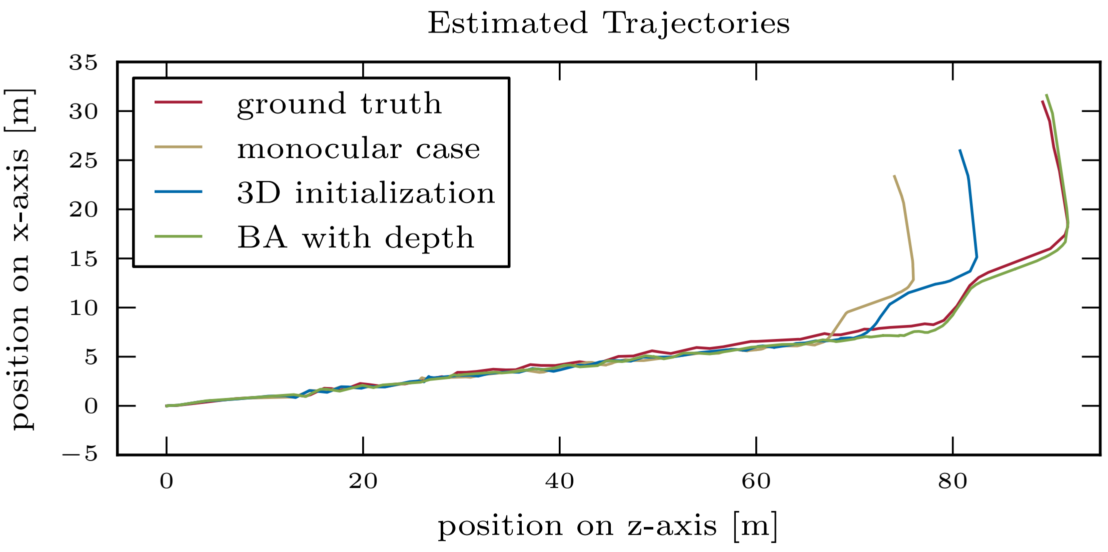


Figure 5.4: Estimated trajectories and ground truth data.

use depth information at the very first keyframe to correctly initialize metric scale of the map at the beginning. Monocular SLAM always requires some additional information if metric position estimates are desired. Initializing the map at the beginning is often feasible, e.g. because we often have some information about the environment close to the start position of a robot (e.g. a landing pad as in Yang *et al.* (2013a)). Without any further metric measurements, however, we have to expect the scale of position estimates and the map will drift over time. This effect is clearly in visible figure 5.4 where we can see how scale is severely underestimated towards the end of the trajectory.

The easiest measure of fighting scale drift with depth measurements is using depth information to initialize new map points. This results in the blue trajectory shown in figure 5.4: It is slightly better, but still suffers from noticeable drift in scale. This is because depth is not used in bundle adjustment: Even though many map points are initialized with their correct scale, bundle adjustment can still arbitrarily change the global scale without any effects on 2D reprojection errors. During optimization, metric scale is still a free variable.

If we also use our proposed combination of 2D and depth measurements in bundle adjustment, we finally arrive at a much more accurate result shown in green in figure 5.4: There is no visible scale drift and the position estimates are very close to ground truth.

The final error at the end of the run using either method is also shown in table 5.1. Using the combination of techniques described in this chapter, we managed to bring the position error down from 16.78 m to 2.13 m, which is equivalent to 2.01 $\%$ drift in translation.

	position error [m]	orientation error [°]
monocular case	16.78	2.61
3D initialization	9.64	2.28
depth constraints	2.13	3.30

Table 5.1: Localization error at the end of each run.

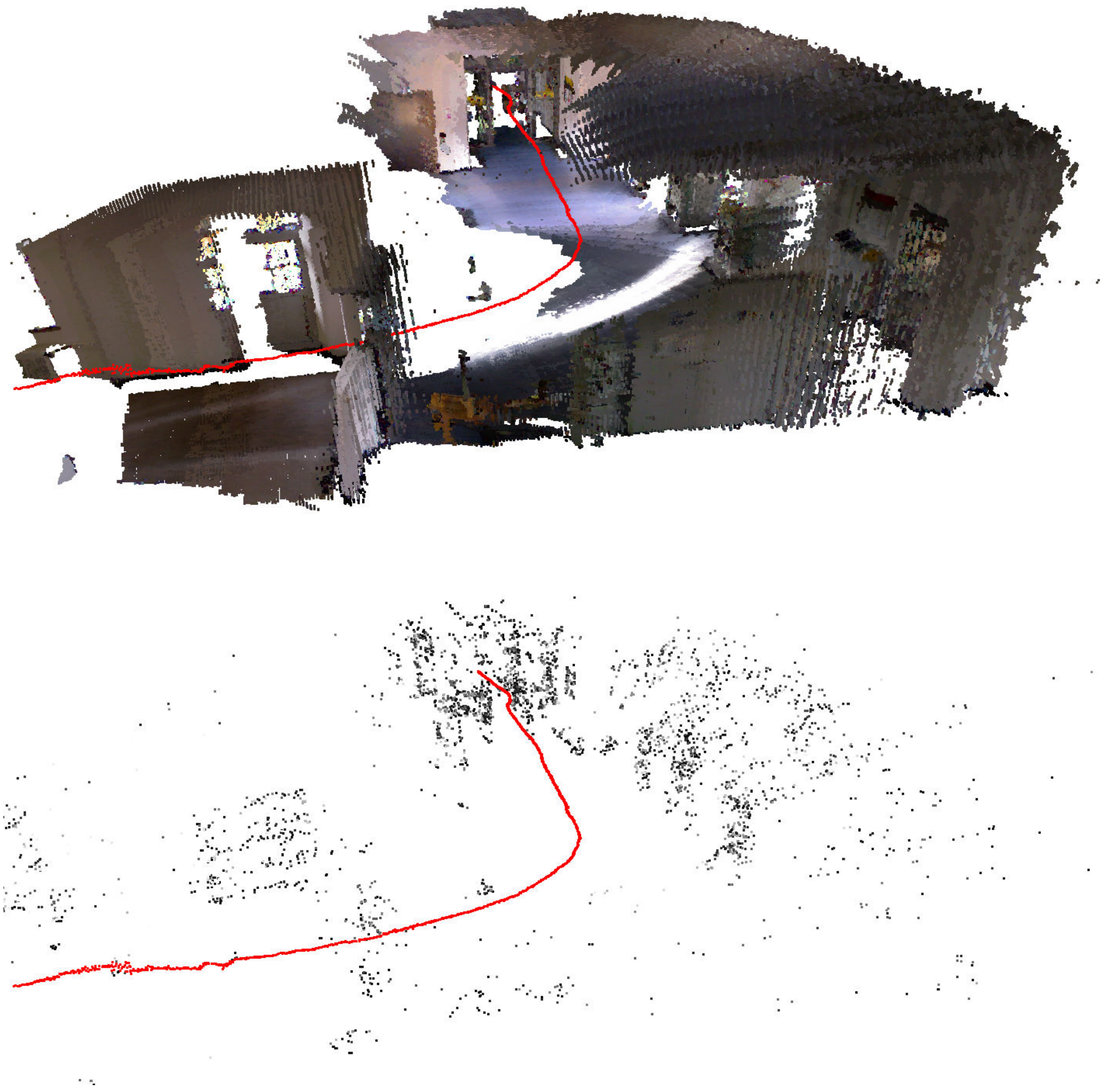


Figure 5.5: One part of the environment (museum room) passed by the wheeled mobile robot during our experiments. Top: Full point cloud reconstructed from registered RGBD images. Bottom: Sparse map from the same perspective, only actual map points used for localization are shown.

5.8 Relation to Stereo Vision

Stereo cameras in conjunction with a stereo matching algorithm are very similar to RGBD sensors: They also provide depth estimates for a subset of image pixels and uncertainty of depth estimates theoretically also grows proportionally to the squared depth. Since they do not rely on an active projector but require natural features visible in both camera images for successful matching, however, matching results are often not as accurate as with structured light often used in depth cameras.

Using our modified version of PTAM with dense stereo data is straighforward, since it can be considered equivalent to RGBD data with a slightly higher depth uncertainty factor. Since we rely on depth estimates only at sparse locations of interest points anyway, however, it is enough to compute sparse stereo matches, which can conveniently be performed in real-time without the help of a GPU Schauwecker *et al.* (2012b). The combination of efficient sparse stereo matching and the method described in this chapter enabled autonomous flight of a quadrotor helicopter relying on stereo cameras as its main sensor as demonstrated in Schauwecker *et al.* (2012a) and Schauwecker and Zell (2014).

5.9 Conclusion

In this chapter we presented a novel method of utilizing RGBD data for visual SLAM. It is based on augmenting monocular visual SLAM by utilizing depth measurements of map points whenever they are available. We implemented our proposed solution by extending the widely-used monocular visual SLAM Parallel Tracking and Mapping (PTAM) and demonstrated that our modifications allow accurate localization and mapping.

Some limitations inherent to PTAM still remain: Its mapping thread tries to run local and global bundle adjustment alternately, but for maps as big as the ones shown in our experiments, the global bundle adjustment step will not be able to finish in any amount of time reasonable for real-time operation, which is why we just disabled it.

This means that PTAM cannot close loops when revisiting places for a second time, since this would require optimizing the full SLAM graph. The authors of PTAM were well aware of this fact but did not care since their intended application was augmented reality in small workspaces and thus they did not even bother to detect loops. PTAM as a software system is thus somewhere in between SLAM and visual odometry: It does not solve the full SLAM system including detecting and closing loops, but it does more than just visual odometry since existing map point positions are improved indeed if they are detected again.

These limitations, among others, lead to our further work described in chapters 6 and finally 9, which do present full SLAM systems.

Chapter 6

Efficient Onboard RGBD-SLAM for Autonomous MAVs

In this chapter we describe a computationally inexpensive RGBD-SLAM solution taylored to the application on autonomous micro aerial vehicles (MAVs). It should enable our exprimental quadrotor MAV to fly in previously unknown environments and create maps of its surroundings completely autonomously, with all computations running on its onboard computer.

We achieve this by implementing efficient methods for both tracking its current location with respect to a heavily preprocessed previously seen RGBD image (keyframe) and efficient relative registration of a set of keyframes using bundle adjustment with depth constraints as a front-end for pose graph optimization. We prove the accuracy and efficiency of our system based on a public benchmark dataset and demonstrate that the proposed method enables our MAV to fly autonomously. The methods and experiments described in this chapter were previously published in Scherer and Zell (2013).

6.1 Related Work: RGBD-SLAM for MAVs

We provided a survey of SLAM methods using RGBD cameras in general in section 5.2 already. Here, we focus on the application of RGBD-SLAM to Micro Aerial Vehicles (MAVs) for autonomous navigation.

Notable cases of autonomous MAVs using RGBD cameras include Huang *et al.* (2011), in which a MAV uses its RGBD camera to compute visual odometry onboard and mapping is done on an external computer using an extension of the system described in Henry *et al.* (2010). While this is impressive work, it is not a fully autonomous MAV according to our previous definition.

An RGBD camera mounted on a quadrotor MAV is used for indoor exploration in Shen *et al.* (2011), which is an interesting topic on its own. Pose estimates, however, are in this case provided using a laser range finder also mounted in the MAV.

6.2 Motivation

We wanted to implement an RGBD-SLAM system that should enable a fully autonomous MAV. By fully autonomous we mean that the SLAM system needs to compute pose estimates that are accurate and fast enough to enable autonomous flight while all computations have to be performed on its onboard computer with limited processing power. For a full SLAM system, this also includes detecting and handling closed loops, which is not possible with PTAM alone, as we saw in chapter 5.

We also wanted to try an alternative tracking method that does not rely on finding interest points in every camera image as is the case for PTAM. This reliance on interest points in every camera image can lead to problems due to few interest points found in images with weakly textured regions and motion blur.

6.3 Software Architecture

We implemented from scratch a keyframe-based RGBD-SLAM system mainly focused to be used on autonomous MAVs. Its main requirements are being able to generate pose estimates at camera rate that are accurate enough to enable autonomous flight, and a scalable mapping process that should be able to map environments not as limited in size as is the case for PTAM.

We adopt one of the major ideas of PTAM, namely separating the two tasks of tracking (localization) and mapping in two individual threads: The time-critical tracking thread is responsible for computing pose estimates wheras the mapping thread builds and refines the map in parallel, which is not as time-critical. As opposed to PTAM, however, we rely on a relative representation of the map, which is detailed as follows.

6.3.1 Map

The most straight-forward map representation for applying bundle adjustment involves storing keyframe poses and map point positions both with respect to a fixed reference frame (see 4.4.4).

In this case, however, we use a *relative* map representation inspired by Sibley *et al.* (2009) and Sibley *et al.* (2010). This means that every map point belongs to one keyframe, in our case the one in which it was seen first, which we will call its *source* keyframe. Map point positions are then stored relative to its source keyframe pose instead of the fixed reference frame.

Even though this relative representation is slightly more complicated, it allows the easy combination of bundle adjustment, i.e. optimizing the full SLAM graph and pose graph optimization, which does not directly affect map points positions.

The map is essentially stored as a set of keyframes that contain both a number of own map points and measurements of map points belonging to other keyframes. In addition to keyframes and map points, we also store a number of pose graph edges connecting pairs of keyframes.

6.3.2 Localization: Tracking Thread

The tracking thread is responsible for processing incoming RGBD image pairs and producing pose estimates at camera rate. For each new RGBD pair, it will compute a prior pose estimate based on a motion model of the camera, decide on the most promising keyframe within the map for tracking, project all map points of this best keyframe according to the pose estimate predicted by the motion model, find the map points close to their predicted location within the current image, and finally estimate the camera pose relative to the reference keyframe using robust nonlinear least squares optimization. If localization is successful, it also needs to decide whether it is time to add the current RGBD pair as a new keyframe to the map.

Motion Model

We implemented a modified version of the decaying-velocity motion model used in Klein and Murray (2007). It assumes the camera to keep moving at a nearly constant but slowly decaying velocity. Given a 6D velocity estimate $\boldsymbol{v}_t \in \mathfrak{se}(3)$ and the previous inverse camera pose ${}^{C_{t-1}}\boldsymbol{T}_R \in \mathrm{SE}(3)$ where C_{t-1} is the camera frame and R a fixed reference frame, it predicts the inverse camera pose at time t as:

$$
{}^{C_t}\boldsymbol{T}_R = \exp(\boldsymbol{v}_t \cdot \Delta t) \cdot {}^{C_{t-1}}\boldsymbol{T}_R \tag{6.1}
$$

Where $\exp : \mathrm{SE}(3) \mapsto \mathfrak{se}(3)$ is the exponential map between the lie group $\mathrm{SE}(3)$ and its lie algebra $\mathfrak{se}(3)$ with $\log : \mathfrak{se}(3) \mapsto \mathrm{SE}(3)$ its inverse. It follows from equation (6.1) that the velocity estimate should ideally be:

$$
\boldsymbol{v}_t = \log\left({}^{C_t}\boldsymbol{T}_R \cdot {}^{C_{t-1}}\boldsymbol{T}_R^{-1}\right)/\Delta t = \log\left({}^{C_t}\boldsymbol{T}_{C_{t-1}}\right)/\Delta t \tag{6.2}
$$

Of course we cannot use ${}^{C_t}\boldsymbol{T}_R$, since this is what we want to predict. We can only use relative poses ${}^{C_{t-1}}\boldsymbol{T}_{C_{t-2}}$ and older to approximate this velocity:

$$
\boldsymbol{v}_t = \alpha_1 \left[\alpha_2 \log\left({}^{C_{t-1}}\boldsymbol{T}_{C_{t-2}}\right)/\Delta t + (1-\alpha_2)\boldsymbol{v}_{t-1}\right] \tag{6.3}
$$

With $\alpha_1, \alpha_2 \in (0,1)$, this is a basic exponentially weighted moving average filter with an additional decay factor α_1 such that it tends to underestimate velocities, which is preferrable to sometimes overestimating velocities.

With relative tracking, the reference frame R is not a fixed world frame but the reference keyframe used for tracking. Its pose itself might be changed by the mapping thread or we might switch to using a different reference keyframe between two successive images. In short, R depends on the time t and is not constant. The prediction step when using relative tracking is thus:

$$^{C_t}\boldsymbol{T}_{R_t} = \exp(\boldsymbol{v}_t \cdot \Delta t) \cdot {}^{C_{t-1}}\boldsymbol{T}_{R_{t-1}} \cdot {}^{R_{t-1}}\boldsymbol{T}_{R_t} \tag{6.4}$$

The relative transform $^{C_t}\boldsymbol{T}_{C_{t-1}}$ between two successive camera poses required for estimating the camera velocity in equations (6.2) or (6.3) is now:

$$^{C_t}\boldsymbol{T}_{C_{t-1}} = {}^{C_t}\boldsymbol{T}_{R_t} \cdot {}^{R_t}\boldsymbol{T}_{R_{t-1}} \cdot {}^{C_{t-1}}\boldsymbol{T}_{R_{t-1}}^{-1} \tag{6.5}$$

The relative transform between both references has to be recomputed at the moment it is needed based on the latest pose estimates of both:

$$^{R_t}\boldsymbol{T}_{R_{t-1}} = {}^{R_t}\boldsymbol{T}_W \cdot {}^{R_{t-1}}\boldsymbol{T}_W^{-1} \tag{6.6}$$

Reference Keyframe Selection

Since we use only one out of all map keyframes for tracking the camera pose, it is important to use the most promising one. In our case, we want to select the reference keyframe which could lead to the most map points measured within the current image. We compute this by projecting all map points of a candidate keyframe into the current image according to the prior pose estimate determined using the motion model and counting the number of points which would lie within the image boundary. This is computationally expensive when testing a large number of keyframes so we only consider candidates within a certain distance to the prior pose estimate in both translation and orientation.

Finding Map Points

After prediction and keyframe selection, can project all map points of the reference keyframe into the current image and need to find their actual location, which is most likely close to where we would expect it.

PTAM does this by considering FAST corners in the neighborhood of the expected position of a map point and computing the zero-mean sum of squared differences (ZMSSD) between image patches around both. This is a trade-off between speed and risking inaccurate or missed matches: Detecting FAST corners is computationally inexpensive and the number of ZMSSD computations required is low, but the actual map point position might not produce a FAST corner in the current image, e.g. due to motion blur, unfortunate lighting or low contrast.

We instead rely on sparse optical flow using the Lucas-Kanade method (see Lucas and Kanade (1981)) using the implementation in OpenCV Bouguet (2001) as it requires finding interest points in keyframes only and can cope with at least limited motion blur in images in between. Sparse optical flow will always succeed to find a *local* minimum and

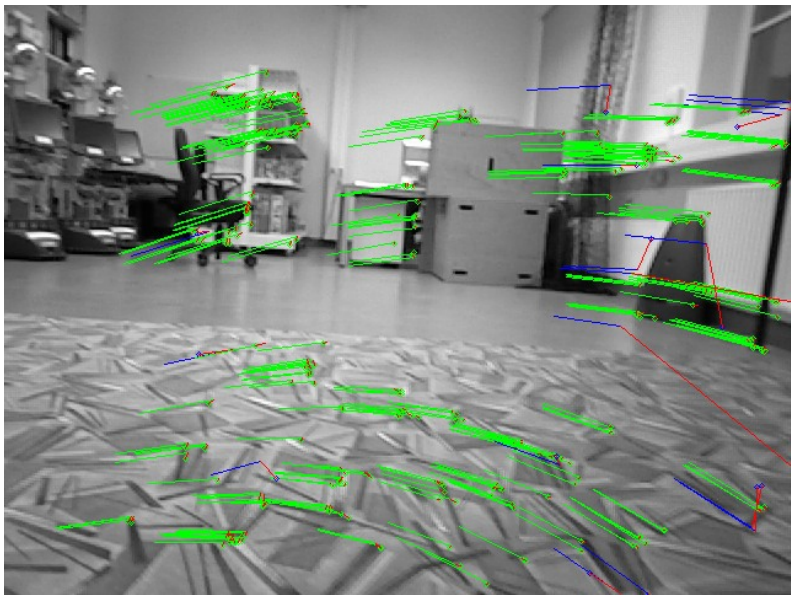


Figure 6.1: Illustration of tracking using sparse optical flow: Green and blue lines are disparities predicted based on the motion model, red lines are corrections from sparse optical flow. Tracks classified as inliers are marked in green, outliers in blue.

can thus in general not be applied to wide-baseline matching. When properly initialized using a motion model as described above, the distances between expected and actual image position of map points are typically very small and wrong matches being found rather unlikely. An example result of this tracking method is shown in figure 6.1. We can clearly see that the correction steps of predicted positions obtained using sparse optical flow (shown in red) for inliers (shown in green) are rather small. The few longer steps belong to measurements later classified as outliers (shown in blue).

Relative Pose Estimation

With the methods described so far, we get a collection of map points with their 3D position known relative to a reference keyframe seen at 2D image positions determined by tracking. Determining the pose of the camera relative to the reference keyframe in this case corresponds to the Perspective-n-Point (PnP) problem as described in Gao *et al.* (2003).

Tracking using sparse optical flow, similar to other methods, in general yields a small number of wrong matches. We identify and exclude outliers using a preemptive RANSAC scheme (see Nistér (2005)) with Gao's solution to the P3P problem (see Gao *et al.* (2003)) to generate candidate hypotheses. Having removed all gross outliers and estimated a hypothesis for the solution with a large consensus set, we employ robust nonlinear least squares optimization to further refine the best hypothesis. In addition to 2D image coordinates only, as used for PnP, we also consider depth measurements at the corresponding image locations, if available, for the final refinement. This optimization is described in more detail in section 6.3.4.

Deciding About New Keyframes

From time to time, the tracking thread should decide to add a new keyframe. We would like the map to consist of enough keyframes such that there is always a possible reference keyframe with a good amount of map points visible in the current frame. We achieve this by adding keyframes whenever the distance to the closest keyframe is too high, when the mean distance in 2D of tracked map points is too high, or when the number of visible map points of the best keyframe is too low.

6.3.3 Mapping

The mapping thread is responsible for combining all incoming keyframes to an accurate and consistent map. It creates keyframes from RGBD pairs, refines pose estimates between new keyframes and their reference keyframe, tries to match new keyframes with more old keyframes, and finally runs pose graph optimization to keep the whole map consistent.

Keyframe Creation

The most important aspect of keyframe creation is selecting map points. We use FAST Rosten and Drummond (2006) corners on multiple pyramid levels. As also noted in Schauwecker *et al.* (2012b), FAST corners tend to flock to image regions with high contrast whereas there might be some lower-contrast areas with no corners at all due to the global threshold. For reliable tracking, we want map points to be evenly distributed among all parts of the image of a keyframe. This is achieved by dividing the image into a grid of $n \times n$ cells. We intentionally set the FAST threshold too low to compute too many interest point and keep the best m *within each grid cell* according to their Harris score. This process is illustrated in figure 6.2.

Relative Pose Refinement

When adding an RGBD image pair as new keyframe to the map, we already have a pose estimate relative to its reference keyframe from tracking. This pose estimate from tracking is derived from forward matches only, i.e. map points of the reference keyframe found in the latest keyframe. After keyframe creation, we also use backward tracking, i.e. try to find map points of the new keyframe in its reference keyframe, as described in section 6.3.2, to find more correspondences. With this higher number of matches, we apply bundle adjustment with depth constraints to further refine the relative pose between

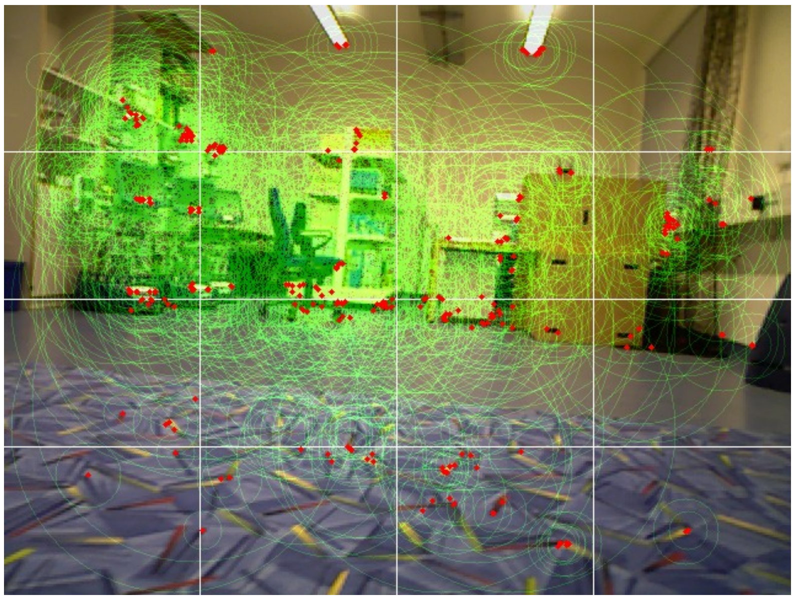


Figure 6.2: Keyframe creation illustrated: FAST corners on various pyramid levels are depicted as green circles. White lines separate grid cells for map point selection. Red dots mark FAST corners actually used as map points.

both keyframes. This optimization is described in detail in section 6.3.4. After bundle adjustment is done, we update positions of map points involved in the optimization and add the relative pose as an edge between both keyframes to the pose graph.

Finding and Utilizing More Reference Keyframes

The pose graph constructed by adding edges to reference keyframes is only a (spanning) tree. For accurate mapping results, we need to be able to add more edges, ideally also closing long loops. We do this by trying to localize the latest keyframe also with respect to the $n_{ref} > 1$ best other keyframes (according to the measure proposed in section 6.3.2), excluding its reference.

Extra care has to be taken to prevent bad edges from being added to the pose graph, since pose graph optimization in general is not robust to outlier edges. Even though there is some work on applying robust methods to pose graph optimization (e.g. Sünderhauf and Protzel (2012)), there often simply are not enough inlier edges to correct for a single outlier edge. We apply both forward and backward tracking for each candidate keyframe. If the two keyframes are k_i and k_j, this yields two relative pose estimates ${}^{i}\boldsymbol{T}_j$ and ${}^{j}\boldsymbol{T}_i$. An additional pose graph edge is only added if both forward and backward tracking consent, i.e. ${}^{i}\boldsymbol{T}_j \approx {}^{j}\boldsymbol{T}_i^{-1}$ after further refinement using bundle adjustment as described in section 6.3.4.

Global Map Optimization

We use HOG-MAN (see Grisetti *et al.* (2010a)) to optimize the pose graph consisting of keyframes and edges between keyframes from their mutual registration. Pose graph optimization is performed after each newly added keyframe, unless the mapping thread is interrupted before.

6.3.4 Optimization with Depth Constraints

We integrate depth measurements as introduced in Scherer *et al.* (2012) and already mentioned in chapter 5. For both refining the solution to a modified perspective-n-point problem with depth in tracking and for registering two keyframes, using relative bundle adjustment with depth.

Perpsective-n-Point Problem with Depth

We use the same combination of 2D and depth reprojection errors for solving the PnP Problem as described in section 5.4.2, except map point positions are now represented relative to their reference keyframe pose. The reprojection errors are thus:

$$\boldsymbol{e}_{2D,i} = h\left({}^{C}\boldsymbol{T}_{R}{}^{R}\boldsymbol{p}\right) - {}^{C}\boldsymbol{u_i} \tag{6.7}$$

$$e_{d,i} = \begin{cases} \left({}^{C}\boldsymbol{T}_{R}{}^{R}\boldsymbol{p}\right)_z - {}^{C}d_i & \text{if } {}^{C}d_i > 0 \\ 0 & \text{else} \end{cases} \tag{6.8}$$

Where ${}^{C}\boldsymbol{u_i}$ is the measured image location in pixels at which the map point was found and ${}^{C}d_i$ is the depth value measured at this location, which is 0 if no depth could be determined by the sensor. For this work, we implemented the optimization using the ceres solver[1], an open source c++ library for robust nonlinear least squares problems described in Agarwal *et al.* (2014).

Both errors are measured in different unrelated units and exhibit different standard deviations as shown in section 5.4.2. We also expect to measure at least some outliers, especially in depth measurements which are not removed using RANSAC, which is why we apply a robustifier kernel ρ. We chose the Huber loss function for this. The objective function minimized by ceres to find the optimal ${}^{C}\boldsymbol{T}_{R}$ is thus:

$$\sum_i \rho\left(\frac{\boldsymbol{e}_{2D,i}}{\sigma_{2D}^2}\right) + \rho\left(\frac{\boldsymbol{e}_{d,i}}{\sigma_d({}^{C}d_i)^2}\right) \tag{6.9}$$

[1]`https://code.google.com/p/ceres-solver/`

Relative Bundle Adjustment with Depth

For relative bundle adjustment we also use depth measurements as described in sect. 5.4.2. The reprojection errors look slightly different due to the relative representation, however. When refining the relative pose of two keyframes, we only consider *relevant* map points p_i, i.e. map points of both keyframes that were also measured in the other keyframe. When considering only two keyframes, each map point $\boldsymbol{p}_i$ can lead to up to 4 different reprojection errors:

- We always measure its 2D position both in its source keyframe at pixel location ${}^{S}\boldsymbol{u_i}$ and in the other at ${}^{O}\boldsymbol{u_i}$, which leads to the measurements $\boldsymbol{e}_{src,2D,i}, \boldsymbol{e}_{oth,2D,i}$ below,
- we might measure its depth in its source keyframe ${}^{S}d_i$ which leads to depth error $e_{src,d,i}$ below,
- we might measure depth in the other keyframe ${}^{O}d_i$ which leads to $e_{oth,d,i}$ below.

$$\boldsymbol{e}_{src,2D,i} = h({}^{S}\boldsymbol{p_i}) - {}^{S}\boldsymbol{u_i} \tag{6.10}$$

$$\boldsymbol{e}_{oth,2D,i} = h({}^{O}\boldsymbol{T}_W {}^{S}\boldsymbol{T}_W^{-1} {}^{O}\boldsymbol{p_i}) - {}^{O}\boldsymbol{u_i} \tag{6.11}$$

$$\boldsymbol{e}_{src,d,i} = \begin{cases} \left({}^{S}\boldsymbol{p_i}\right)_3 - {}^{S}d_i & \text{if } {}^{S}d_i > 0 \\ 0 & \text{else} \end{cases} \tag{6.12}$$

$$\boldsymbol{e}_{oth,d,i} = \begin{cases} \left({}^{O}\boldsymbol{T}_W {}^{S}\boldsymbol{T}_W^{-1} {}^{S}\boldsymbol{p_i})\right)_3 - {}^{O}d_i & \text{if } {}^{O}d_i > 0 \\ 0 & \text{else} \end{cases} \tag{6.13}$$

The overall objective function minimized using ceres is:

$$\sum_i \rho\left(\frac{\boldsymbol{e}_{src,2D,i}}{\sigma_{2D}}\right) + \rho\left(\frac{\boldsymbol{e}_{oth,2D,i}}{\sigma_{2D}}\right) + \rho\left(\frac{\boldsymbol{e}_{src,d,i}}{\sigma_d({}^{S}d_i)}\right) + \rho\left(\frac{\boldsymbol{e}_{oth,d,i}}{\sigma_d({}^{O}d_i)}\right) \tag{6.14}$$

6.4 Experiments and Results

6.4.1 Evaluation: Benchmark Dataset

We first evaluate the described system on the file *fr3long_office_houshold* from the benchmark dataset described in Sünderhauf and Protzel (2012) and available online.[2] The two main objectives are computational efficiency and accuracy of pose estimates, so we evaluate both at the same time.

[2] `http://vision.in.tum.de/data/datasets/rgbd-dataset`

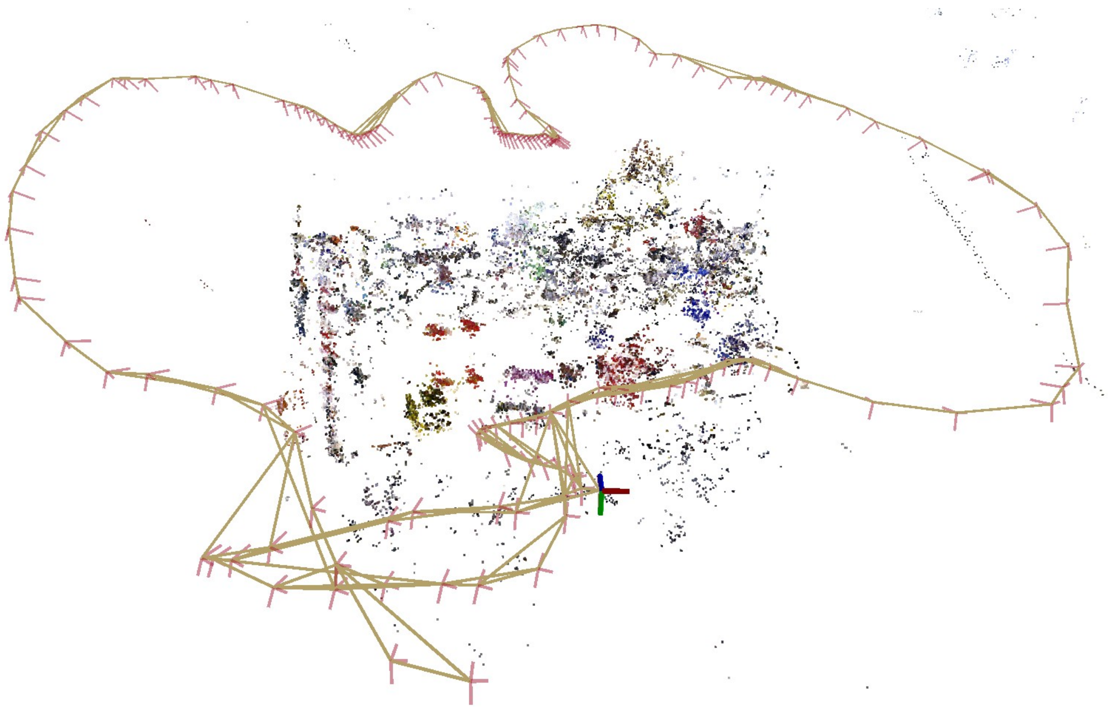


Figure 6.3: Visualization of the map obtained by applying our proposed system on the freiburg3 dataset: Keyframe poses (red), edges between keyframes (golden), and map points (in their original color).

Computational Efficiency

We measured the time required for each individual step on two different computers: A laptop computer with an Intel Core 2 Duo P9400 CPU running at 2.40 GHz and the on-board single-board computer used on our MAV with an Intel Core 2 Duo SL9400 Low Voltage running at 1.86 GHz. Measured mean times and standard errors of steps performed within the tracking thread for each camera image are shown in table 6.1, steps performed for each new keyframe by the mapping thread are shown in table 6.2. We can see that tracking is clearly fast enough to process frames faster than 30 Hz or 33 ms, even on the slightly slower on-board computer. The mapping thread has to perform some longer and more complex computations, which does not pose a problem, since they are only required for new keyframes. The mapping thread can be interrupted early if a new keyframe is dropped, theoretically as soon as the first step of keyframe creation is done, which is so fast that it should never have to be interrupted by a new keyframe.

In our experiments, however, we only interrupted the mapping thread after refinement using bundle adjustment was done, but before checking for and adding additional pose graph edges.

task	laptop [ms]	SBC [ms]
sparse optical flow	6.38 ± 1.06	8.38 ± 1.23
preemptive RANSAC	3.80 ± 0.46	5.23 ± 0.28
robust optimization	1.90 ± 0.93	2.44 ± 1.18
total	12.08 ± 1.46	16.05 ± 1.62

Table 6.1: Computation times required for localization (tracking thread).

task	laptop [ms]	SBC [ms]
keyframe creation	12.54 ± 2.02	15.60 ± 2.20
reverse tracking	13.05 ± 4.16	16.41 ± 1.32
refinement (BA)	32.96 ± 15.20	43.62 ± 19.27
additional edge accepted	51.95 ± 12.50	67.46 ± 15.74
additional edge rejected	22.11 ± 2.08	30.03 ± 2.63
pose graph optimization	16.20 ± 9.61	21.70 ± 12.66

Table 6.2: Computation times required for processing a new keyframe (mapping thread).

Localization Accuracy

We used the same dataset for accuracy evaluation with the tool included in the benchmark dataset. It reports an absolute position root mean square error (RMSE) of 13.6 cm comparing the full estimated trajectory to ground truth. Since we perform both tracking and refinement relative to reference keyframes, we are mainly interested in position and orientation errors relative to the corresponding reference keyframe. This is shown in table 6.3. Note that errors for tracking and mapping are not directly comparable since transforms estimated in refinement typically consist of slightly bigger translations and rotations compared to the relatively small transforms estimated in tracking. A visualization of the resulting map with keyframes, edges of the pose graph, and map points obtained by running our SLAM system on the benchmark dataset is shown in figure 6.3.

method	tracking	refinement (BA)
position RMSE	$1.8\,\mathrm{cm}$	$1.4\,\mathrm{cm}$
position MAE	$1.2\,\mathrm{cm}$	$1.1\,\mathrm{cm}$
orientation RMSE	$0.95°$	$0.85°$
orientation MAE	$0.67°$	$0.60°$

Table 6.3: Mean absolute errors (MAE) and root mean square errors (RMSE) of pose estimates relative to reference keyframe on the freiburg3 dataset.

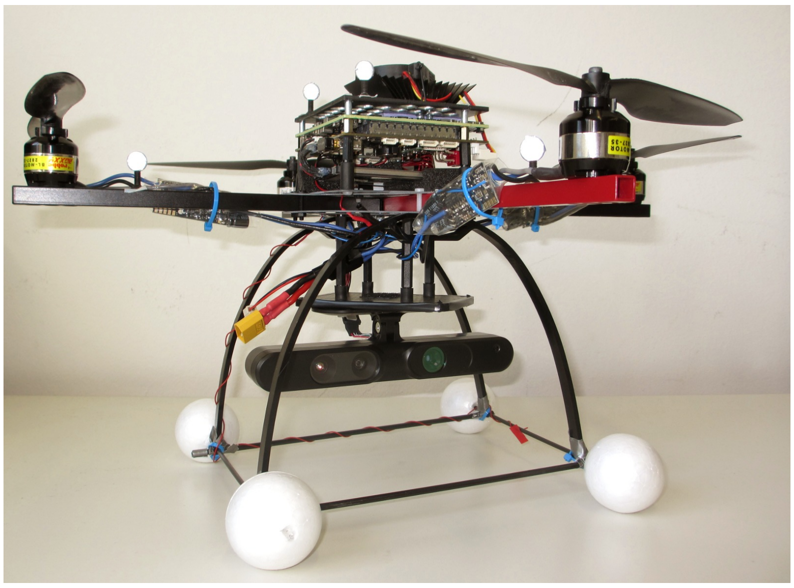


Figure 6.4: The MAV used for this work

The camera trajectory starts at the keyframe at the bottom and ends where the colored pose marker is. It is interesting to see that our system is accurate enough to implicitly close the loop in this case, without explicit loop closure detection.

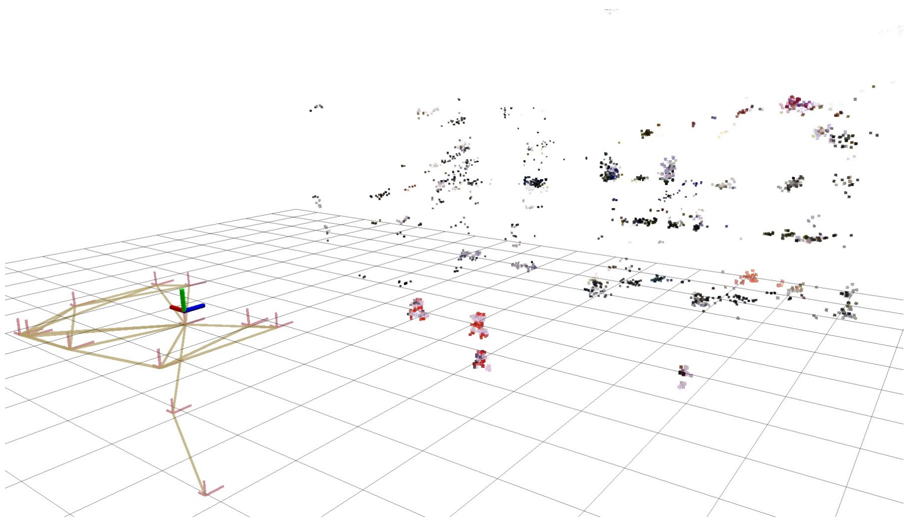
6.4.2 Evaluation: Autonomous Flight

Experimental Setup

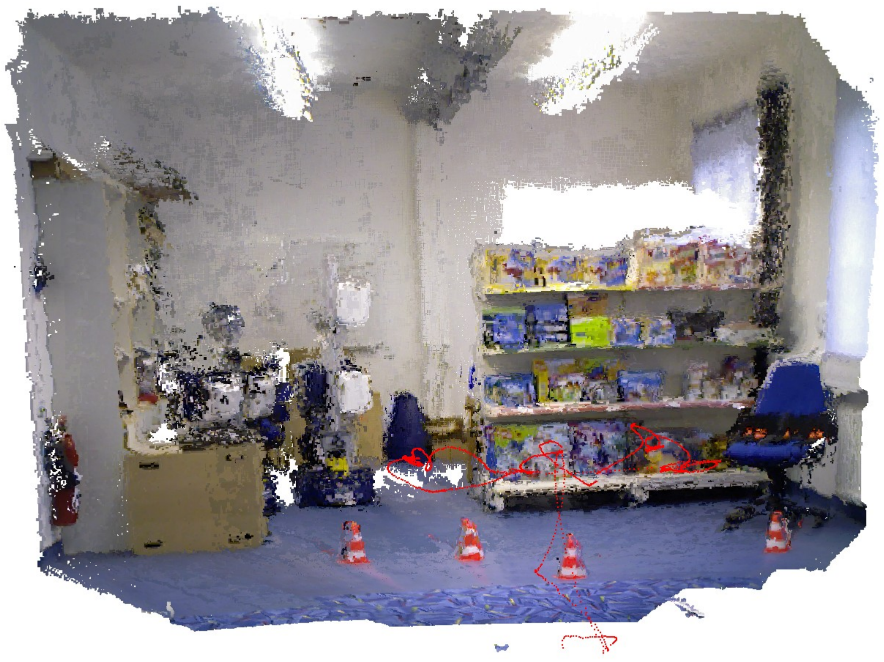
We used the same system running in real-time on the on-board computer of our MAV shown in figure 6.4 while it was flying autonomously in our laboratory. We commanded it to follow a path consisting of predefined way-points on a rectangular pattern. Navigation was performed by the nested PD controller implemented on the pxIMU autopilot. We commanded the MAV to fly autonomously and compared the pose estimates with ground truth data captured by an external tracking system.

Accuracy

We used an Optitrack external tracking system (ETS) by Naturalpoint consisting of 7 V100:R2 cameras, which provides pose estimates at a rate of 100 Hz and used this as ground truth to again compute relative errors in position and orientation for both tracking and refinement. The resulting errors are shown in table 6.4. They are slightly higher than before, which is to be expected due to the more difficult image sequence with abrupt motions especially during takeoff and landing, which also introduces considerable motion blur. The map built during this flight is illustrated in figure 6.5a.



(a) Map and pose graph built during autonomous flight.



(b) Dense point cloud and trajectory reconstructed from recorded data in off-line processing after the flight.

Figure 6.5: Mapping result obtained from autonomous flight.

method	tracking	refinement (BA)
position RMSE	1.7 cm	2.5 cm
position MAE	1.5 cm	2.3 cm
orientation RMSE	1.71°	1.00°
orientation MAE	1.30°	0.79°

Table 6.4: Errors on the autonomous MAV dataset.

6.5 Conclusions

In this chapter we presented a computationally very efficient RGBD-SLAM system which is able to run in real-time on the on-board computer of our autonomous MAV. This is achieved by a combination of fast tracking and localization relative to a single keyframe and bundle adjustment with depth constraints as SLAM front-end, generating keyframe-to-keyframe constraints for pose graph optimization, performed by a SLAM back-end.

By the time of the original publication in Scherer and Zell (2013), this was the first time a MAV was shown to perform RGBD-SLAM with global pose graph optimization on its on-board computer. As we will see later (in table 9.1), the accuracy of pairwise keyframe registration as visual odometry is not as accurate as using local bundle adjustment as in chapter 5. This work can still be considered an important stepping stone towards the final SLAM system described in chapter 9.

Chapter 7

Loop Closure Detection Using Depth Images

The work in this chapter explores the use of depth alone. The general idea is that all variants of RGBD-SLAM trade off reliance on RGB (or intensity) on one hand and depth information on the other hand. Our own work presented in chapters 5 and 6, for example, relies mostly on intensity images and requires only few depth measurements for correct scale and improved accuracy. Dense methods as briefly described in section 5.2.2, on the other hand, mostly rely on depth information and optionally RGB (or intensity). Methods relying on depth images alone, however, usually lack a method of explicitly detecting closed loops, which is essential for a full SLAM system capable of large-scale mapping.

There are, however, a plethora of attempts to visually detect closed loops based on regular (intensity) camera images (e.g. Cummins and Newman (2008), Konolige and Agrawal (2008), Zhang (2011)). The general consensus about the approach chosen for visual loop closure detection seems to contain first finding the previous images that are most similar to the current image (based on a content-based image retrieval scheme) and afterwards discarding those that cannot be loops due to heuristics or geometry consistency checks. The currently most commonly used image retrieval scheme is Bag of Visual Words (see Sivic and Zisserman (2003)).

There are some competing approaches to Bag of Words for loop closure detection that rely on using raw image descriptors (e.g. Zhang (2011)) or locality-sensitive hashing (LSH, e.g. Shahbazi and Zhang (2011)). In both cases, the authors report to obtain much better results using alternative approaches than using Bag of Words. We are not so sure how representative that comparison is, however, due to very small datasets used and the fact that Bag of Words was used with very small vocabulary sizes: at most 2500 words were used in Shahbazi and Zhang (2011) and at most 8000 in Zhang (2011), as opposed to 1 million in e.g. Nister and Stewenius (2006).

The work described in this paper started as a bachelor's thesis (Kloss (2012)) and was subsequently extended and published in Scherer *et al.* (2013).

7.1 Local Features

We evaluate various interest point detectors and local descriptors for depth images. Some algorithms contain both interest point detection and local descriptor extraction, e.g. Normal Aligned Radial Features (NARF, see Steder *et al.* (2011)) or the Scale-Invariant Feature Transform (SIFT, see Lowe (2004)). In general, there are two classes of algorithms: Those which were developed for depth images exclusively and those which are adaptations of known algorithms for intensity images.

7.1.1 Interest Points

Interest points or keypoints are image locations in regions that contain a relatively high amount of information (i.e. texture in intensity images). They are usually desired to be distinctly located, which is typically the case for local extrema or corners, repeatable, and robust to changes of perspective. We here review three promising keypoint detectors for depth images: The Harris corner detector, FAST features, and the NARF interest point detector.

Harris

The Harris corner detector (Harris and Stephens (1988)) is a well-known interest point detector for intensity images. It considers the variance of the intensity values around each pixel. For a corner and therefore a stable interest point, intensity variation in both x and y direction needs to be relatively high, which corresponds to both eigenvalues of the so called *Harris matrix* H of the patch having large positive values. The response C is usually computed without explicitly determining the eigenvalues of H:

$$H = \begin{bmatrix} \widehat{I_x^2} & \widehat{I_x I_y} \\ \widehat{I_x I_y} & \widehat{I_y^2} \end{bmatrix} \tag{7.1}$$

$$C = |H| - k \cdot \text{trace}(H) \tag{7.2}$$

Here I_x denotes the x-component of the intensity gradient and $\widehat{\cdot}$ is a shorthand for the weighted sum over a certain influence window. C above is the so-called Harris response. An alternative response measure is the Shi and Tomasi response: $C = \min(\lambda_1, \lambda_2)$, where λ_i are the eigenvalues of H. When using depth images instead of intensity images, the image gradients are roughly proportional to the x and y components of the surface normals.

Keypoints detected by the Harris detector tend to lie directly on the boundaries of objects. This works fine in continuous 2D images but can easily become a problem with depth images, since the border regions are often unstable with regard to both surface normal estimation and depth measurement.

FAST

Similar to Harris, the FAST keypoint detector (Rosten and Drummond (2006)) was originally developed for intensity images. Its main idea is to compare intensities of pixels on a Bresenham circle of 16 pixels diameter around each keypoint candidate p to the intensity of p itself. If the patch around p contains a corner, there should be at least $n = 12$ consecutive pixels on that circle that are either all darker or all brighter than p by some threshold t. The order of pixel tests is optimized such that non-corners can be eliminated as soon as possible. Instead of using FAST on intensity values, we can also easily ”abuse” it and apply it to depth values.

NARF

NARF (see Steder *et al.* (2011)) interest point detection begins with border detection, where borders are defined as “non-continuous traversals from foreground to background”. If a point has a border in its neighbourhood it is assigned the direction of the border as its dominant direction of surface change, otherwise the first principal component of the curvature is used. The interest value of a point is then influenced by two aspects:

- The interest value of a point is decreased if there are points with strong surface changes nearby, in order to encourage keypoints on stable surfaces.
- Points with pairs of neighbors with different directions of surface have their interest values increased.

As a result, NARF interest points can be found close to object boundaries but not directly on these boundaries, which is usually good for depth images or point clouds.

7.1.2 Local Descriptors

Local descriptors are designed to describe an image patch (usually located around an interest point) in a compact but distinctive way. Similar to the location of interest points, a point’s descriptor is usually desired to be invariant to changes in scale and perspective.

NARF

The main idea of the NARF descriptor is similar to 2D algorithms like SIFT: For each keypoint, a normal aligned range value patch is computed that is orthogonal to the surface normal of the point. Changes in depth values are computed along 36 directions around the keypoint and weighted by their distance to it. A unique orientation is obtained by computing a direction histogram over the descriptor values. The bin with the maximum value is selected as the dominant orientation and the descriptor can be made rotational invariant by rotating (shifting) the histogram by this orientation. It should be noted that the algorithm will compute multiple descriptors at a single keypoint if there is more than one bin within the histogram with values exceeding 80% of the maximum. Because of this, NARF will often produce more descriptors than there are keypoints for a given image.

Kernel Descriptors

Bo *et al.* (2010) introduce a kernel view of SIFT and Histograms of Oriented Gradients (HOG, see Dalal and Triggs (2005)) features and demonstrate that comparing HOG descriptors can be interpreted as computing a linear match kernel that combines two subkernels comparing gradient orientation and magnitude of all pixel pair combinations. They propose a slightly different orientation kernel and add a Gaussian position kernel to arrive at what they call gradient kernels. The problem of these kernel features is the fact that they are generally infinite-dimensional. Bo et al. propose sampling basis vectors over a fine grid to obtain finite-dimensional features. These are still too high-dimensional to be of any use, so a kernel PCA is used to compact features to a 200-dimensional feature space. They show that this notion of kernel descriptors can also be applied to come up with completely new descriptors, e.g. color or shape kernels and kernel descriptors can successfully be applied to depth images.

BRIEF

Binary Robust Independent Elementary Features (BRIEF, see Calonder *et al.* (2012)) are very simple descriptors in the form of a binary string. BRIEF was developed for intensity images to allow for fast computation, efficient storage and also fast comparison by using the Hamming distance instead of the common L_2 norm. In order to obtain the descriptor, the intensity values of several point pairs in the neighborhood of a keypoint are compared after smoothing the patch to reduce noise. Each descriptor entry is then either 1 or 0, depending on which one of the two points had the higher intensity. One would think that the performance heavily depends on how well the comparison pattern is chosen. As the experiments in Calonder *et al.* (2012) showed, however, truly random (but of course constant) patterns outperformed both symmetric and manually chosen regular comparison patterns.

7.2 Bag of (Visual) Words

Bag of Visual Words (BoW), also referred to as Bag of Keypoints (Csurka *et al.* (2004)), is a technique widely used in computer vision to compute a single global descriptor of an image, given an arbitrary number of descriptors of local features found within this image. The general idea is rooted in document processing, where documents of arbitrary length can be represented using a global descriptor by counting occurrences of a finite number of n keywords (i.e. the vocabulary). The resulting histograms, which we will call BoW vectors from now on, can be interpreted as an n-dimensional global descriptor of a document and efficiently compared using any vector norm.

In computer vision, representative feature descriptors are used as visual words. A visual vocabulary, i.e. a set of representative feature descriptors, can be computed from a large set of example features by clustering, typically k-means clustering. Computing the BoW vector of a given image consists of extracting all local features, determining the corresponding visual word for each feature (e.g. using fast linear approximate nearest neighbour search), and finally counting the occurrence of each visual word.

The utility of Bag of Visual Words heavily depends on the vocabulary size. Nister and Stewenius (2006) propose vocabularies that contain on the order of millions of visual words and claim that using such large vocabularies, image retrieval works accurately even without considering the geometric layout of visual words. In order to cope with vocabulay sizes this large, they suggest hierarchical vocabulary trees: During vocabulary creation, k-means clustering is performed recursively on multiple levels up to a maximum tree depth, clustering the set of all sample features that belong to one cluster in the previous level in turn. Looking up the corresponding visual word for a given feature involves traversing the vocabulary tree down to a leaf node, finding the closest representative descriptor on each level.

7.3 Loop Closure Detection

Since the main focus of this work is evaluating different *features* with respect to their application for loop closure detection, we employ only a very basic loop closure detection method. For each image, it will do the following steps:

1. Query the database of previous images to find and return the most similar ones using Bag of Words as described in section 7.2.
2. Disregard all resulting candidates that belong to the 10 latest keyframes in order to prevent detecting loops already when the robot is still at the same location.
3. Compute the similarity s of the current image with each resulting candidate based on their BoW vectors v_1, v_2 and disregard those whose similarity is below a user-chosen threshold α. We use the similarity measure described in Galvez-Lopez and Tardos (2012), which is based on the L_1 norm:

$$s(v_1, v_2) = 1 - \frac{1}{2}\left|\frac{v_1}{|v_1|_1} - \frac{v_2}{|v_2|_1}\right|_1$$

4. Return the up to k images with the highest similarity as loop closure candidates, where k can be chosen by user. The choice of k will depend on how many candidates can be further verified by checking the geometric consistency within a reasonable amount of time.

5. Finally, the current image is added to the database so it can later be found as a loop closure candidate.

One could think of many more heuristics to filter out more false positives in loop closure detection. Galvez-Lopez and Tardos in Galvez-Lopez and Tardos (2011) propose many more heuristics, e.g. enforcing temporal consistency, i.e. requiring detection of the same loop multiple times in a row before it is actually reported.

The final step within a real SLAM system, however, should always be a geometric consistency check. This could involve trying to register loop closure candidates (e.g. using ICP) and determining their goodness of fit, or matching local features within a RANSAC scheme and counting the number of inliers. Also evaluating registration methods, however, would go beyond the scope of this work, so we evaluate the performance of the returned loop closure detection candidates without checking their geometric consistency.

7.4 Implementation

We implemented a highly modular (polymorphic) and easily configurable loop closure detector in C++. The user can choose among any of the interest point detectors and local descriptors mentioned in sect. 7.1 and arbitrarily set all of their parameters. Adding more interest point detectors or local descriptors is easy and only involves writing one more derived wrapper class, which will register with the corresponding object factories.

We use the DBoW2 library described in Galvez-Lopez and Tardos (2012), a very efficient open-source implementation of the hierarchical vocabulary tree approach mentioned in sect. 7.2.

For SIFT, FAST, and BRIEF, we rely on their implementations found in OpenCV[1]. For Harris and NARF, their implementations found in PCL[2] are used. As the C-Version of Kernel Descriptors[3] only allows the computation of dense kernel descriptors, i.e. sampled over a grid of overlapping patches, we modified the source code to allow computation of sparse kernel descriptors at given keypoint locations.

7.5 Benchmark Dataset

In order to evaluate the performance of loop detection algorithms, we need sequences of depth images with ground truth pose information in environments as diverse as possible. To our knowledge, there is currently only one applicable dataset publicly available, published in Sturm *et al.* (2012), which features sequences of RGBD images with ground truth 6D pose information obtained by an external tracking system. The number of loops

[1]http://opencv.org/
[2]http://pointclouds.org/
[3]http://www.cs.washington.edu/ai/Mobile_Robotics/projects/kdes/

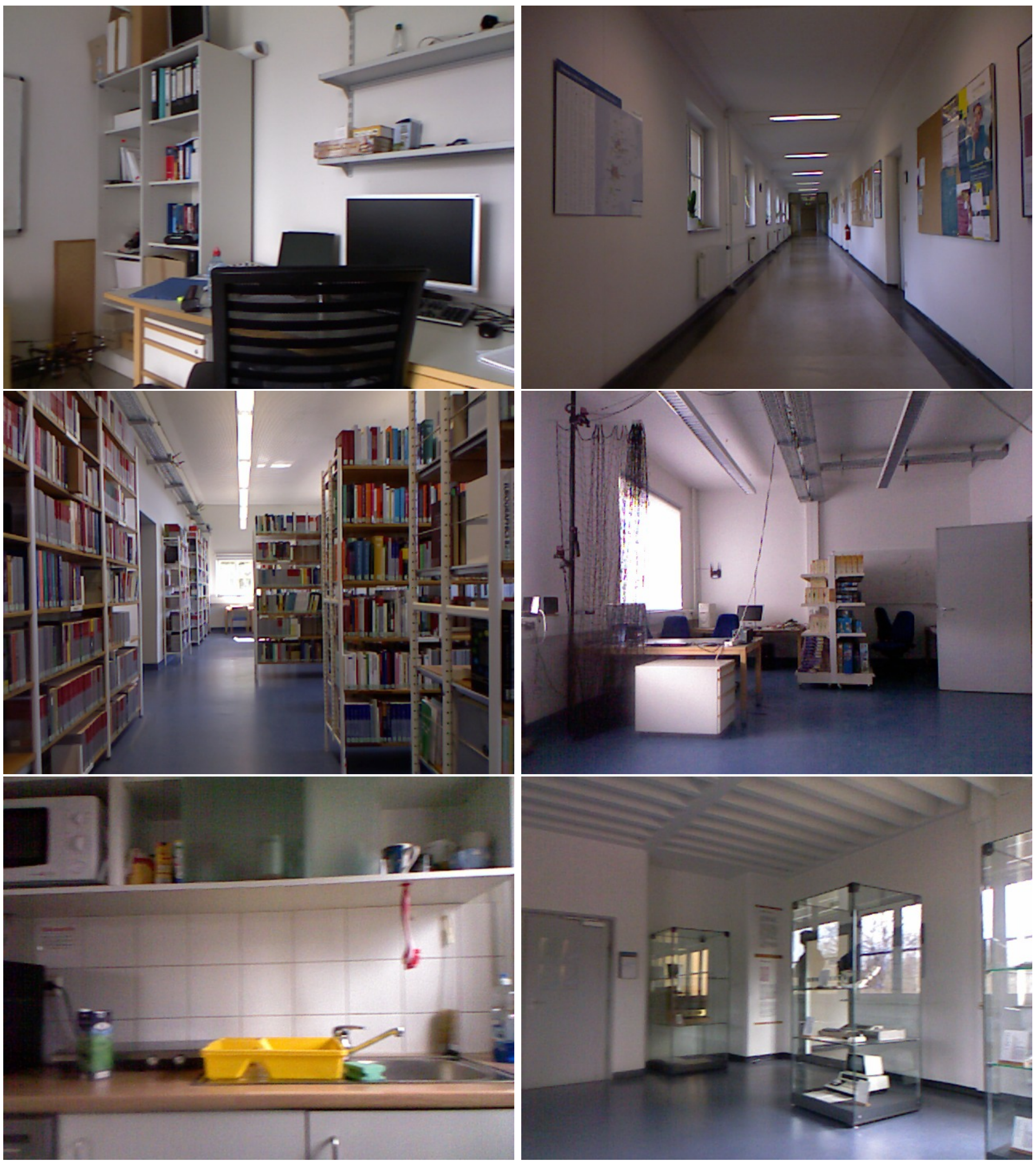


Figure 7.1: Examples of different scenes encountered by the robot.

in this dataset, however, is not high enough for a proper evaluation of loop closure detection, so we additionally recorded our own data.

We used a Scitos G5 robot by Metralabs with a forward-looking Microsoft Kinect mounted on its top. Its 2D pose is obtained by localizing it within a previously built map using odometry and its laser range finder. We manually drove this robot on multiple loops through various environments available in our building. Our final set of sequences consists of:

- 5 runs within our robots laboratory and on the corridor just outside of it,
- 4 runs within our department's library,
- one run in which the robot enters different offices several times
- one run within our department's computer museum,
- one run within our kitchen,
- and 4 runs containing loops taken from the Freiburg dataset[4].

Examples of the various views encountered by the robot are illustrated in figure 7.1 Instead of keeping full sequences of all these runs, we significantly reduce the number of images to a subset of keyframes, as it is usually sufficient to find loops for new keyframes only in keyframe-based SLAM.

7.6 Experiments and Results

7.6.1 Evaluation

Determining True Loops

In order to evaluate the performance of the loop closure detection the ground truth must be established first. Depending on the dataset, poses with three or six degrees of freedom were given for each image. We determined whether two images were taken at the same place by considering the euclidean distance of the translation and the angle between the orientations of both poses: Two images are counted as a loop if their translation is less than $2\,\mathrm{m}$ and their angle is less than $30°$.

As we allow a distance of up to $2\,\mathrm{m}$ between two images belonging to a loop, it must be ensured that the images were not taken on the same visit to this place. Therefore, there must be at least 10 keyframes between two images in order for them to be considered a loop.

Evaluating Classification Performance

At first glance, loop closure detection is a binary classification problem: Given a place described by an image and a database of images, decide whether the place has been visited before. Therefore it seems natural to evaluate the performance in terms of *sensitivity*

[4]We used the logfiles named "freiburg1_360", "freiburg1_room", "freiburg2_large_with_loop", and "freiburg3_long_office_household"

(SE) and *specificity* (SP) and e.g. calculate a ROC curve (*receiver operating characteristic*). The problem at hand, however, is not only to decide whether a place has been visited before, but also to identify the place correctly by retrieving a matching image from the database. This makes the definition of *true positives* (TP), *false positives* (FP), *true negatives* (TN) and *false negatives* (FN) more difficult: If a place has been visited before and no image from the database is retrieved, it is a clear *false negative*. If a correct image is found it is obviously a true positive. But if a wrong image is retrieved for a previously visited place this could be counted as *false positive* (the retrieved image was not taken at the current position) as well as *false negative* (as the loop was not detected correctly). We called these cases *wrong positives* (WP) and decided to count these towards the *false negatives*, as we consider an undetected loop a graver mistake than a false loop candidate. The reason for this is simply that false loop candidates could be filtered out by further checks as discussed in section 7.3 whereas a missed loop is final. A diagram of the definitions can be seen in Fig. 7.2. Based on these numbers, we can compute the sensitivity (SE) and specificity (SP) in the following way:

$$SE = \frac{TP}{TP + FN^*}$$
$$SP = \frac{TN}{FP + TN}$$

Where we apply the sum of both, false negatives and wrong positives, instead of the classical false negative count:

$$FN^* = FN + WP$$

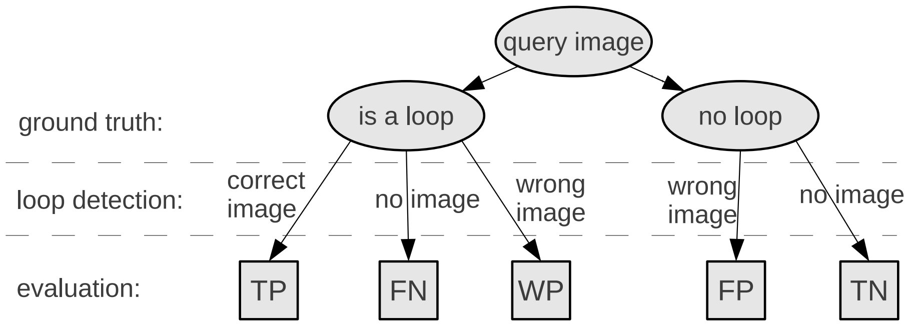


Figure 7.2: Classification of loop candidates into categories for evaluation.

7.6.2 Parameter Optimization

The performance of each of the algorithms employed in loop closure detection (i.e. interest point detectors, local descriptors, bag of words, and the actual decision about loop closure detection) heavily depends on a certain number of parameters. Finding universally good values for these parameters is often impossible as the optimal values depend on the problem, the input data provided to the algorithm, and how its output is further processed. To allow a fair comparison, however, we need to find the best parameter values for the problem at hand, i.e. test the full system including vocabulary creation and loop closure detection for various parameter values.

Vocabulary creation involves k-means clustering, which is usually implemented using a randomized algorithm. In order to get meaningful results, we run it at least five times with the same parameters and compute the mean over all runs.

This means we need to evaluate our loop detection system on the benchmark dataset thousands of times, when a single run can take up to a few hours for computationally expensive local features. We utilized grid computing on a cluster of 8 nodes with 16 CPU cores each to run up to 128 evaluations in parallel.

Conceptually, the task of loop closure detection using bag of words consists of two major steps. The first step involves using local features and bag of words to suggest loop closure candidates. In this first step, our main objective is a high sensitivity: If there is a loop, we want it to be among the suggested candidates. In a second step, the actual loop closure detection is performed based on the candidates obtained in the first step. Here, we want to obtain a good trade-off between specificity and sensitivity by tuning the receiver operator characteristic (ROC) curve.

Vocabulary size

We first try to determine a good vocabulary size using default parameter values of various methods used. The optimal vocabulary parameters of NARF and FAST interest points with kernel descriptors were a *branching factor* (i.e. the number of clusters during each clustering step) of $b = 20$ and *level* (i.e. the maximum tree depth of the resulting vocabulary tree) of $l = 3$, which corresponds to an efficient vocabulary size of $20^3 = 8000$. Using FAST with BRIEF descriptors, the optimal vocabulary size is smaller with $b = 5$ and $l = 5$, whereas the optimal vocabulary size of SIFT is orders of magnitude bigger with $b = 20$ and $l = 4$. Since we, however, wanted to focus on depth features and use a consistent vocabulary size, we decided to use $b = 20$ and $l = 3$ for further experiments, which seems to work well for all methods.

Reference: SIFT on Intensity Images

Since our benchmark dataset consists of RGBD image pairs with both color and depth images, we first try to detect loops using SIFT features computed on the intensity images as a reference.

We tested various values for the parameters:

- *contrast threshold* $ct \in \{0.01, 0.02, 0.04, 0.08\}$
- *edge threshold* $et \in \{2, 5, 10, 15\}$
- *sigma* $\sigma \in \{1.2, 1.4, 1.6, 2.0, 2.5\}$

and found that $ct = 0.01$, $et = 10$ and $\sigma = 2.5$ obtained the best sensitivity of 94.6%.

FAST & BRIEF

FAST has only one parameter (its threshold) and we decided to keep BRIEF's amount of bytes constant at 32 bytes. We thus only had to vary one parameter: The best FAST threshold $t \in \{6, 8, 10, 12, 14, 16, 18, 20, 22\}$ turned out to be $t = 6$, which resulted in the best sensitivity of 87.5%. The choice of the lowest value for the threshold t does not come as a surprise, since this basically means it will consider a rather high number of interest points.

NARF

NARF features require the user to choose a large number of features, which makes large-scale search on the full high-dimensional grid computationally intractable due to the combinatorial explosion of the number of possible combinations. We decided to always enable rotation invariance and fix the angular resolution $ar = 0.3$, as this appeared to be the highest resolution for which we can still compute NARF features within a reasonable amount of time. We evaluated various values for the the following parameters:

- support size: $ss \in \{0.1, 0.2, 0.4\}$
- minimum keypoint distance: $mkd \in \{0.125, 0.25, 0.5\}$
- optimal surface distances: $osd \in \{0.125, 0.25, 0.5\}$
- minimum interest values: $miv = \{0.225, 0.45, 0.9\}$
- minimum surface changes: $msc = \{0.1, 0.2, 0.4\}$
- optimal patch size: $ops = \{51020\}$

The grid search lead to the optimal values $ss = 0.1$, $mkd = 0.125$, $osd = 0.125$, $miv = 0.225$, $msc = 0.10$, $ops = 5$, with which we obtained a maximum sensitivity of $SE = 76.1\%$.

Harris Interest Points

We evaluated Harris interest points combined with various descriptors from other methods with their respective best parameters, optimizing the following parameters:

- radius $r \in \{0.1, 0.05, 0.025, 0.01\}$
- threshold $t \in \{0.025, 0.01, 0.005, 0.0025\}$
- method $m \in \{Harris, Curvature, Tomasi, Noble\}$

Using BRIEF descriptors, the best sensitivity 88.1% was achieved for $r = 0.01$, $t = 0.0025$ and $m = Tomasi$. In combination with NARF descriptors, the best sensitivity 79.8% was also obtained for the same values $r = 0.0025$, $t = 0.01$ and $m = Tomasi$.

Sparse Kernel Descriptors

Since kernel descriptors do not provide their own interest point locations, we need to combine these with an interest point detector. Unfortunately, the source code of Kernel Descriptors does not include the functionality to train new kernel descriptors, so we only used the pre-trained ones that come bundled with the source code. They use a fixed patch size of 16×16 pixels. The only parameter we can modify directly is *low contrast*: The gradient magnitude for each image patch is normalized by dividing it by its L2-norm or *low contrast* if it is below *low contrast*. This protects from noise artifacts being amplified too much when there is not much structure. The original source code of kernel descriptors computes these descriptors on heavily downscaled versions of input images. We therefore introduced a new parameter *scale factor* by which images (and keypoints) are scaled before computing kernel descriptors.

We evaluated the performance of kernel descriptors (KDES) in combination with FAST, Harris, and NARF interest points using their respective optimal parameters as determined before. We checked all combinations of the following parameter values:

- $lc \in \{0.75, 0.9, 1.0, 1.1, 1.25\}$
- $rf \in \{0.5, 0.25, 0.125, 0.1, 0.05\}$

In all three cases, the same parameters $lc = 0.9$ and $rf = 0.25$ turned out as optimal, resulting in sensitivities of 89.6% for FAST and Harris interest points and 92.0% using NARF interest points.

Varying the Loop Closure Detection Parameter

Since we are typically not only interested in a high sensitivity but also a high specificity, we evaluated the loop closure detection method described in sect. 7.3 using various

Interest Points	Descriptor	Best Sensitivity
FAST	BRIEF	87.5 %
HARRIS	BRIEF	88.1 %
NARF	NARF	76.1 %
HARRIS	NARF	79.8 %
FAST	KDES	89.6 %
HARRIS	KDES	89.6 %
NARF	KDES	92.0 %
SIFT	SIFT	94.6 %

Table 7.1: Best sensitivities obtained for $k = 3$ loop detection candidates using different interest points and descriptors

values for α. This leads to one receiver operating characteristic (ROC) curve for each interest point detector/descriptor pair, which are drawn in Fig. 7.3.

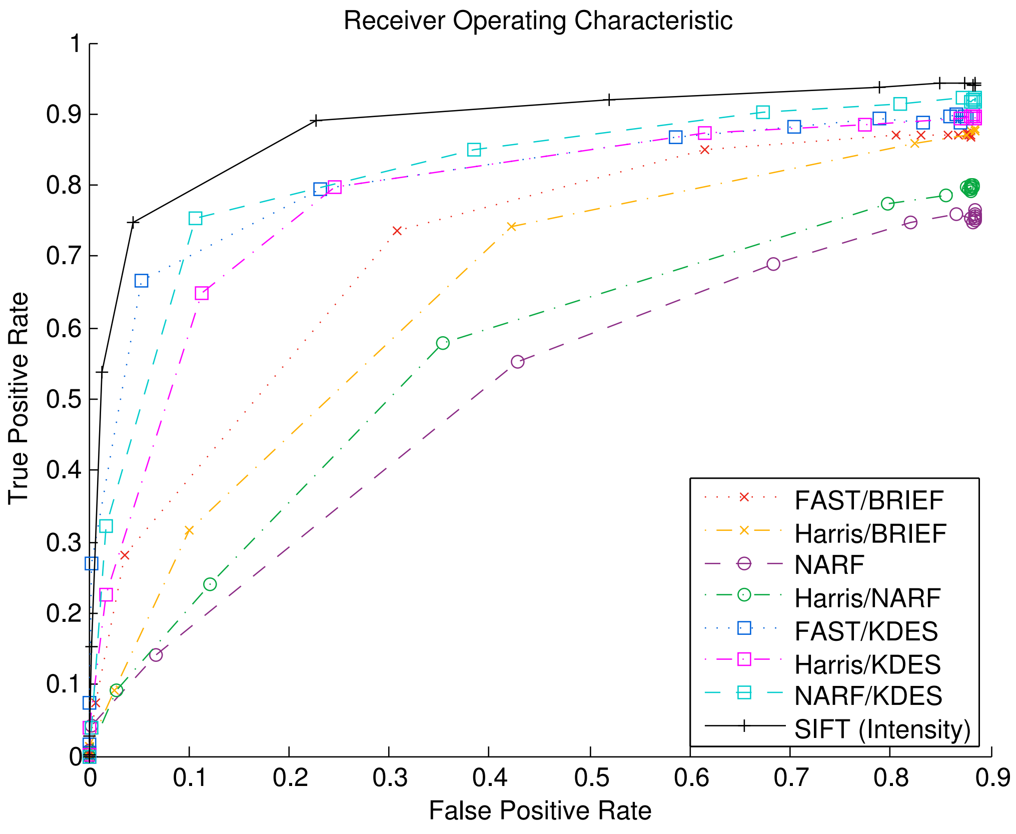


Figure 7.3: Receiver operating characteristic.

Varying the Number of Considered Loop Closure Candidates

In the previous experiments, we considered $k = 3$ loop candidates for each image. If a correct loop is among these k, it counts as detected. The choice of $k = 3$ was made by us rather arbitrarily. In practise this parameter should be chosen depending on a combination of the desired sensitivity and how computationally expensive it is to verify a loop candidate geometrically (e.g. using iterative closest point or based on feature matches). The influence of k on the sensitivity is shown in Fig. 7.4. For this experiment, we ran each method 10 times instead of 5 times to obtain more meaningful estimates of the standard deviation, which is shown using error bars.

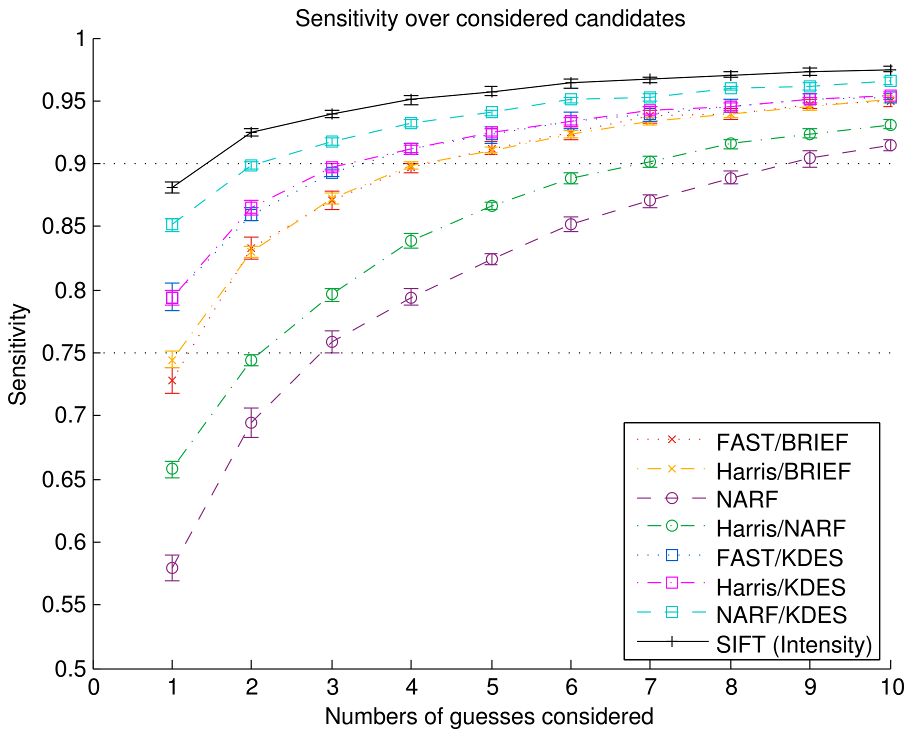


Figure 7.4: Sensitivity and its standard deviation vs. the number of considered loop closure candidates.

7.7 Conclusions

7.7.1 Discussion

The results show that the performance of loop closure detection using depth images with any kind of depth features is always worse than when using intensity of color images. This should not come as a surprise, since intensity images typically contain more information useful for loop closure detection: When the camera is pointed towards a planar wall on a corridor for example, we cannot expect to reliably detect any loop using depth images alone. The intensity image in this case, however, might capture distinct *texture* information, e.g. of a unique poster on the wall, which makes detecting this loop much easier. One such example of this scenario from our dataset is shown in Fig. 7.5.

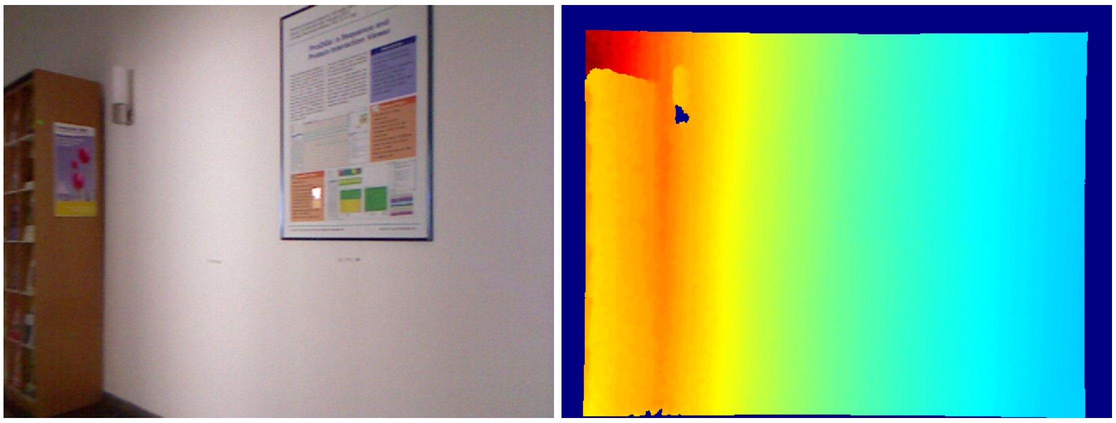


Figure 7.5: RGB and depth image pair from the dataset. The RGB image clearly contains much more relevant information since the highly textured poster is not "visible" to the depth camera.

Considering this drawback, the results also show that loop closure detection using depth images works surprisingly well. Even if we only consider the single most similar image obtained using the best combination of all methods tried, we can find close to 85% of all loops using NARF interest points with kernel descriptors or roughly 73% using the computationally very inexpensive combination of FAST interest points with BRIEF descriptors. This should be more than enough for a typical SLAM setting: When a robot enters a room for the second time, we can expect it to detect one of usually several possible loops eventually.

Considering the performance of the various individual methods tested, we can make the following interesting observations:

The choice of interest points is not as important as the choice of the descriptor

This is especially obvious from Fig. 7.4, where we find the curves of different interest points in combination with the same descriptor to be nearly identical. This is in parts due to how we posed our optimization problem in section 7.6.2: Since we try to find the best sensitivity and do not care about the number of interest points, we usually end up with low thresholds and thus high numbers of interest points. Comparing results using a fixed number of interest points might have been more fair, but enforcing a fixed limit is also not trivial. Also, we usually do not care so much about the number of interest points as long as this does not make it computationally prohibitively expensive.

"Abusing" 2D methods originally intended for intensity images works surprisingly well compared to proper 3D features

The performance of the computationally least expensive combination of FAST interest points and BRIEF descriptors is only surpassed by kernel descriptors with different interest points, but works better than NARF. It appears that for loop closure detection, we do not need the additional information encoded in 3D features like NARF. This can be explained by the fact that true loops in our dataset consist of a pair of images that are captured by roughly the same pose. This means there is usually no considerable change in scale, orientation, or perspective in general. Additionally, regions with invalid depth measurements are usually reproducible, i.e. invalid depth measurements also contain useful information (see Fig. 7.6), whereas 3D features that rely on 3D points to recover surface normals will disregard these regions completely. Finally, 3D methods like Harris and NARF might work rather badly in our dataset because they suffer from many high depth values for which there is considerable depth inaccuracy.

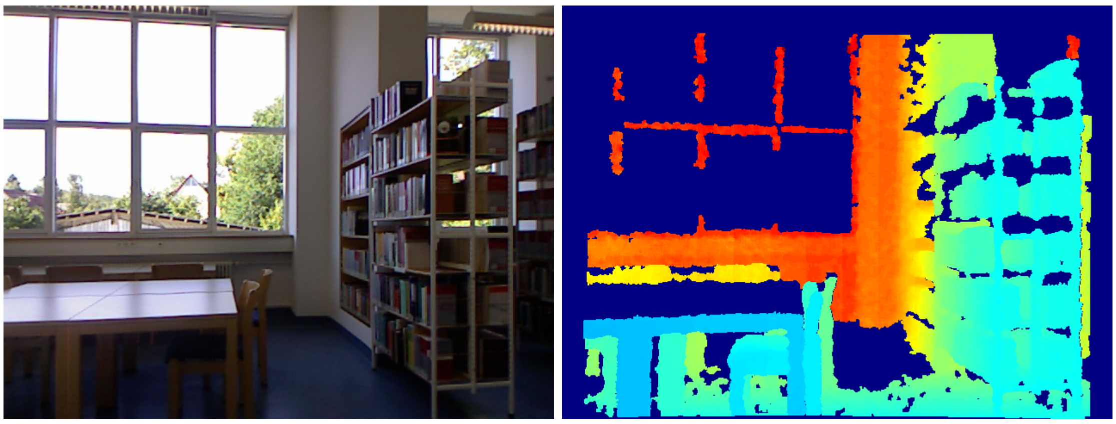

(a) RGB image (b) Depth image

Figure 7.6: Example image pair from our dataset: Borders between valid and invalid depth values contain useful information about the scene.

7.7.2 Summary

We evaluated several methods of interest point detection and descriptor extraction for the task of loop closure detection using bag of words based on depth images. As expected, the achieved sensitivity is lower than what can be obtained on intensity images, but still high enough that it should be usable for SLAM based on depth images alone. It turns out that basic 2D features known from intensity images work surprisingly well for loop closure detection, which might be mainly due to a combination of important information being contained in invalid depth readings and no big changes in scale, orientation or perspective in general for real loops. The combination of two computationally very efficient methods FAST and BRIEF obtains surprisingly good results. It might be possible to combine fast depth registration techniques with loop closure detection to a computationally inexpensive full SLAM system using depth images alone.

Chapter 8

Drift-Corrected Visual SLAM

8.1 Motivation

In the preceding sections, we relied solely on camera images and indirectly estimated their relative poses by means of SLAM. Using these relative pose estimates alone, a robot can estimate its pose only relative to previous poses and the images it recorded there: Be it by directly registering the current image to one previous keyframe as shown in chapter 6 or indirectly by estimating the current pose relative to a set of 3D map points, whose location was in turn triangulated based on a number of previous keyframes as done in PTAM, described in chapter 4.

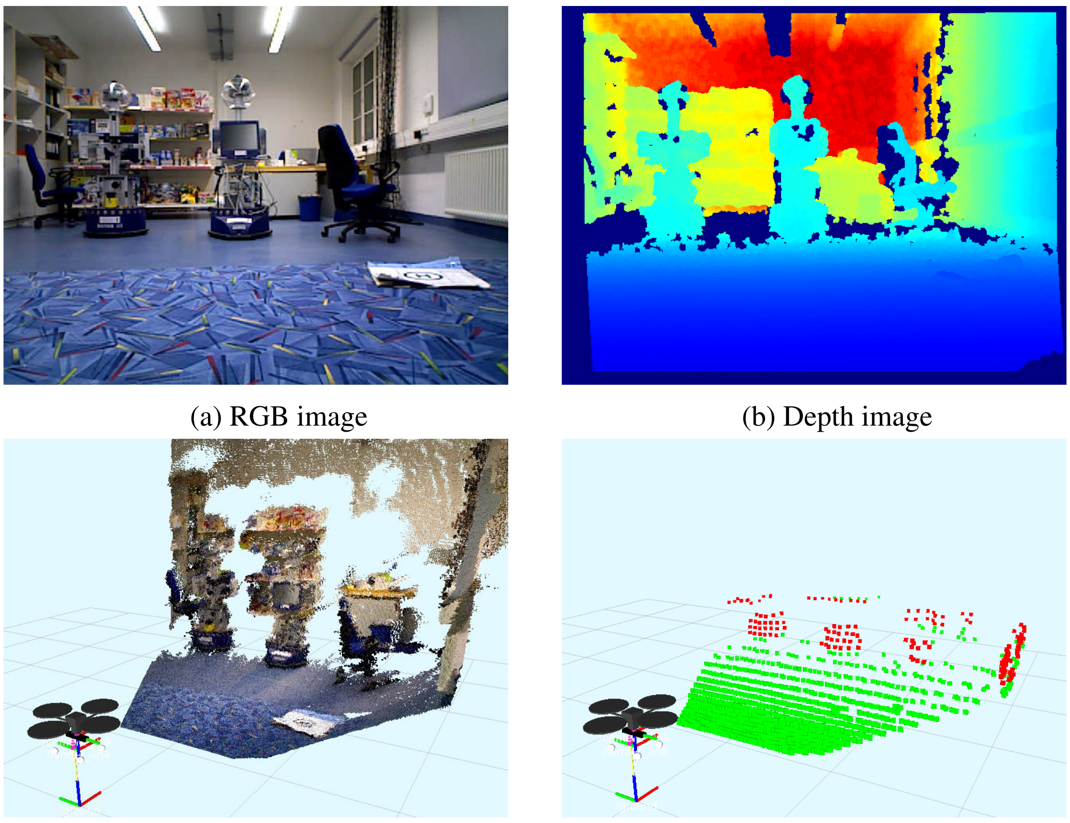
The absolute pose can only be approximated by providing the relative pose estimate with respect to one previous (e.g. the initial) pose, which is to be assumed fixed. Any absolute pose estimate derived in this way has to inevitably drift over longer trajectories, similar to pose estimates inferred based on dead reckoning using odometry measurements from wheel encoders.

Drift can be fatal: It is typically low within a local neighborhood and in SLAM typically consistent with the drift exhibited by the map. For aerial robots, however, drift-free correct attitude and altitude estimation is essential to autonomous flight over longer trajectories.

When navigating an MAV in GPS-denied environments, which is typically either indoors or outdoors but close to the ground, the MAV can often see the floor, most of the time as a prominent ground plane. A sample view is illustrated in Fig. 8.1. The main idea of this chapter is to estimate and utilize this ground plane information for long-term drift correction to allow for longer autonomous flight.

8.2 Ground Plane Detection

During a typical indoor flight, the ground plane can be clearly identified in most RGBD point clouds and thus also depth images seen by the on-board camera, which is demonstrated in Fig. 8.1. We propose a ground plane detection method that consists of the following three steps: Sub-sampling the dense point cloud to use a much smaller number of sparsely reconstructed 3D points, detecting inliers and outliers using a custom



(a) RGB image (b) Depth image

(c) RGBD point cloud (d) Ground plane visualization

Figure 8.1: Typical view of an onboard RGBD camera while navigating in an indoor scenario: (a) RGB image, (b) Depth image, (c) RGBD point cloud and (d) visualization of detected ground plane.

RANSAC scheme, and finally robust refinement of the most promising hypothesis based on inliers.

8.2.1 Sampling 3D Points

We sample the depth image at few sparse locations to significantly reduce the number of depth measurements that need to be considered. We obtain good results even when considering only one depth value out of a 10×10 grid. For a typical non-aggressive flight near hovering with a forward-looking camera, we can usually also disregard the upper half of the image, since the ground plane can be expected in the lower half of the image. Only if there are very few valid depth measurements in the lower half, which might be the case because the MAV is flying very close to the ground, we fall back to considering the full depth image. Using the intrinsic camera calibration, depth pixels are then reprojected to 3D points.

8.2.2 Inlier/Outlier Detection

Out of the still large number of all 3D points determined by sub-sampling, only a minor fraction might actually lie on the ground plane. To make matters worse, there might be multiple prominently visible planes (walls, furniture, tables) that should not be confused with the ground plane. In order to reliably extract the ground plane, we employ a preemptive RANSAC scheme (Nistér (2005)) based on a custom inlier/outlier/even worse outlier model to quickly arrive at a well-supported hypothesis for the actual ground plane.

Inlier/Outlier/Worse Outlier Model

RANSAC (RANdom SAmple Consensus) as originally introduced in Fischler and Bolles (1981) tries to find a hypothesis which maximizes the size of its consensus set. The consensus set here is the set of measurements whose values agree with (i.e. are close enough to) the measurement values as predicted by the hypothesis. The fact that each measurement can either count as an inlier our outlier is also called the inlier/outlier model in Nistér (2005), since it only considers the binary decision whether a measurement is to be considered an inlier or outlier.

Early on in our experiments, however, we found that RANSAC using the basic binary inlier/outlier model often fails in ground plane detection when facing a vertical wall or in front of a wide obstacle. A simplified 2D sketch of such a failure case is shown in Fig. 8.2: Both hypotheses lead to exactly the same numbers of inliers (7) and outliers (2), so it is impossible to identify a better solution by counting inliers and outliers alone.

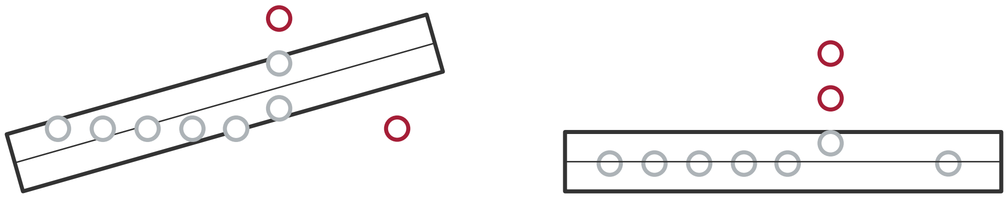


Figure 8.2: (a) A typical failure case of the basic inlier/outlier model for RANSAC in ground plane detection and (b) the desired estimate obtained using our proposed inlier/outlier/worse outlier model.

When trying to find ground planes, however, there are two very different kinds of outliers which are very easy to distinguish: Points above the ground plane are ubiquitous, since the camera sees walls and obstacles protruding from the ground all the time. Points below the ground plane, however, should be very rare: During indoor flight on a single floor of a building, we expect to see none except for some completely random sensor failures.

We thus introduce a novel ternary inlier/outlier/even worse outlier model, which modifies the score of a hypothesis according to the following rules: $+1$ if the observation is an inlier, $+0$ if the observation is a ”good” (i.e. expected) outlier above the ground plane, and $-c_{bad}$ if it is a bad (i.e. unexpected) outlier below the ground plane. We choose $c_{bad} = 10$, which worked well in all of our experiments. In the toy example above, hypothesis (a) would lead to the score $s_a = -3$ and (b) $s_b = 7$. This means our proposed ternary model correctly solves this problem.

8.2.3 Robust Refinement

RANSAC will return a hypothesis which is best with regards to its scoring function, which in turn only depends on the number of inliers and (in our case different kinds of) outliers, but it does not necessarily find the most accurate hypothesis. If accuracy is desired, it is required to further refine this hypothesis using its inliers alone. One popular method of fitting planes to 3D point clouds is using the principal component analysis (PCA) method Weingarten *et al.* (2004).

Given N points $p_i = (x_i, y_i, z_i)^T$, the PCA method assumes the plane to pass through their arithmetic mean:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} p_i \tag{8.1}$$

The normal of the plane should correspond to the smallest Eigenvalue of the so-called scatter matrix, which corresponds to the sample covariance matrix:

$$A = \frac{\sum_{i=1}^{N} (p_i - \mu) \cdot (p_i - \mu)^T}{N-1} \tag{8.2}$$

The decision whether a measurement is to be considered an inlier or outlier is made by comparing the measurement error to a pre-defined threshold. This threshold cannot be chosen arbitrarily low since even inlier measurements are subject to measurement noise. There will thus always be some points erroneously classified as inliers that do not belong to the ground plane but to the lower parts of obstacles close to the ground. It is thus advisable to use a robust or robustified method for this refinement instead of basic least squares.

We propose to use a robustified Campbell (1980) version of the PCA method for plane fitting to refine the rough ground plane estimate obtained using RANSAC based on its inliers. The robustified versions of the mean in equation (8.1) is:

$$\mu = \frac{\sum_{i=1}^{N} w_i \cdot p_i}{\sum_{i=1}^{N} w_i} \tag{8.3}$$

And the robustified version of equation (8.2):

$$A = \frac{\sum_{i=1}^{N} w_i^2 \cdot (p_i - \mu) \cdot (p_i - \mu)^T}{\left(\sum_{i=1}^{N} w_i^2\right) - 1} \tag{8.4}$$

Here, w_i are individual weights calculated for each point. The basic PCA method for plane fitting can be considered a special case of the robustified version that treats all points equally, i.e.

$$w_i = 1 \quad \forall i \tag{8.5}$$

In this case, equations (8.1) and (8.2) are equivalent to equations (8.3) and (8.4).

A robust estimator of mean and covariance is described in Campbell (1980), however, computes the weights w_i based on the Mahalanobis distance of point p_i from the current hypothesis:

$$w_i = \frac{\omega(d_i)}{d_i} \tag{8.6}$$

Where d_i is the Mahalanobis distance between p_i and the current estimate of the mean and ω is an influence function:

$$\omega(d) = \begin{cases} d & \text{if } d < d_0 \\ d_0 \exp\left(-\frac{(d-d_0)^2}{b_2^2}\right) & \text{else} \end{cases} \tag{8.7}$$

Once the ground plane is found and refined, we can represent it using its normal vector pointing up in the current camera frame ${}^{C}\boldsymbol{up}$ and the camera height h with respect to the plane.

8.3 Drift Correction

Given visual odometry pose estimates that drift over time and occasional ground plane detections, we want to combine both of them to a drift-corrected pose estimate such that

- corrected pose estimates are consistent with the latest ground plane estimate, and
- correcting visual odometry pose estimates introduces as few disturbances as possible, i.e. the correction transformation applied to to visual odometry pose estimates should be minimal in translation and rotation.

Using visual odometry alone, we can only estimate the camera pose with respect to a fixed reference frame by incrementally combining all sequential relative pose estimates in a chain (dead reckoning):

$$^{W}\boldsymbol{T}_{C_i} = {^{W}\boldsymbol{T}_{C_0}} \cdot {^{C_0}\boldsymbol{T}_{C_1}} \cdots {^{C_{i-1}}\boldsymbol{T}_{i}} \tag{8.8}$$

Ground plane measurements, on the other hand, put constraints on parts of the absolute pose $^{W}\boldsymbol{T}_{C_i}$ and are shortcuts in the chain above.

Our proposed method of combining both for drift-corrected pose estimates $^{W}\hat{\boldsymbol{T}}_{C_i}$ consists of two steps:

- Prediction using the relative pose estimate inferred from visual odometry:

$$^{W}\bar{\boldsymbol{T}}_{C_i} = {^{W}\hat{\boldsymbol{T}}_{C_{i-1}}} \cdot {^{C_{i-1}}\boldsymbol{T}_{C_i}} \tag{8.9}$$

- Correcting the predicted pose estimate by applying a minimal correction transform such that the resulting pose estimate is more consistent with the measured ground plane:

$$^{W}\hat{\boldsymbol{T}}_{C_i} = {^{W}\bar{\boldsymbol{T}}_{C_i}} \cdot \boldsymbol{T}_{corr} \tag{8.10}$$

Correction Transform Computation

The drift-correction transform $\boldsymbol{T}_{corr}$ consists of rotation component $\boldsymbol{R}_{corr}$ and translation $\boldsymbol{t}_{corr}$. The rotation part should bring the measured up vector in camera coordinates $^{C_i}\boldsymbol{up}$ parallel to the up direction expected based on the current camera pose:

$$^{W}\bar{\boldsymbol{R}}_{C_i} \cdot \boldsymbol{R}_{corr} {^{C_i}\boldsymbol{up}} \overset{!}{=} {^{W}\boldsymbol{up}} \tag{8.11}$$

$$\boldsymbol{R}_{corr} \cdot {^{C_i}\boldsymbol{up}} \overset{!}{=} {^{W}\bar{\boldsymbol{R}}_{C_i}^{-1}} \cdot {^{W}\boldsymbol{up}} \tag{8.12}$$

There is a closed-form solution to compute the shortest rotation $\boldsymbol{R}_{a,b}$ given two unit vectors $\boldsymbol{a}, \boldsymbol{b}$ such that $\boldsymbol{R}_{a,b} \cdot \boldsymbol{b} = \boldsymbol{a}$, which can be derived from the fact that $\boldsymbol{R}_{a,b}$ should rotate around an axis orthogonal to $\boldsymbol{a}$ and $\boldsymbol{b}$ by the angle between both vectors.

$$\boldsymbol{R}_{a,b} = \boldsymbol{I}_3 + \boldsymbol{S} + \frac{1}{1+d} \cdot \boldsymbol{S} \cdot \boldsymbol{S} \tag{8.13}$$

Where $\boldsymbol{S} = [\boldsymbol{a} \times \boldsymbol{b}]_{\times}$ is the cross-product matrix of the cross product and $d = \langle \boldsymbol{a}, \boldsymbol{b} \rangle$ is the dot product of $\boldsymbol{a}$ and $\boldsymbol{b}$. To compute $\boldsymbol{R}_{corr}$ above, we choose $\boldsymbol{a} = {^{C_i}\boldsymbol{up}}$ and $\boldsymbol{b} = {^{W}\bar{\boldsymbol{R}}_{C_i}^{-1}} \cdot {^{W}\boldsymbol{up}}$

The translation component on the other hand, should be a small translation along the plane normal $\boldsymbol{t}_{corr} = s \cdot {^{C_i}\boldsymbol{up}}$ that brings the corrected camera pose to the measured height h above the ground plane:

$$h \overset{!}{=} \left({}^{W}\hat{\boldsymbol{T}}_{C_i}\right)_{3,4} = \left({}^{W}\bar{\boldsymbol{T}}_{C_i} \cdot \boldsymbol{T}_{corr}\right)_{3,4} \tag{8.14}$$

$$\overset{!}{=} \left({}^{W}\bar{\boldsymbol{R}}_{C_i} \cdot \boldsymbol{t}_{corr} + {}^{W}\boldsymbol{t}_C\right)_z \tag{8.15}$$

$$\overset{!}{=} s\left({}^{W}\bar{\boldsymbol{R}}_{C_i} \cdot {}^{C_i}\boldsymbol{up} + {}^{W}\boldsymbol{t}_C\right)_z \tag{8.16}$$

Which can easily be solved for s.

Correction Filtering

Even though ground plane estimates are immune to long term drift, they still suffer from high-frequency measurement errors. Applying the full correction would introduce these errors into the corrected pose estimates. We instead limit the influence of a single attitude measurement by scaling it with a gain factor $\alpha < 1$:

$$\boldsymbol{T}_{corr} = \exp\left(\alpha \cdot \log\left[\begin{pmatrix} \boldsymbol{R}_{corr} & \boldsymbol{t}_{corr} \\ 0 & 1 \end{pmatrix}\right]\right) \tag{8.17}$$

8.4 Experiments and Results

We evaluate the system described above and its individual building blocks using the TUM RGBD benchmark dataset described in Sturm *et al.* (2012), which contains several log files of image streams captured with an RGBD camera, including its ground truth pose estimates obtained from an external tracking system. We choose four logs that correspond to a handheld-slam (all) scenario and ideally contain clearly visible ground planes.

8.4.1 Ground Plane Detection

We first compare the accuracy of the various ground plane detection techniques described in Sect. 8.2 based on the ground truth data included in the file *fr3_long_office_household*. The results can be seen in table 8.1, where *RANSAC* denotes the common preemptive RANSAC method using an inlier-outlier scheme, *I/O/W RANSAC* is preemptive RANSAC using our Inlier/Outlier/Worse Outlier Model, I/O/W + PCA is the above followed up with a refinement step using the PCA method and *I/O/W + robust PCA* uses our robust refinement method. We compute height error Δh and attitude error $\Delta\alpha$, which is the angle between actual and expected up direction. We can clearly see that each extension successively improves the result. Plane estimates using basic RANSAC are more than one order of magnitude worse compared to all other methods because it often detects other planar surfaces (e.g. the surface of a desk) instead of the actual ground plane.

	$\Delta\alpha$ in $[^\circ]$		Δh in $[\mathrm{cm}]$	
	MAE	RMSE	MAE	RMSE
RANSAC	25.25	33.22	33.35	35.38
I/O/W RANSAC	1.23	1.53	2.60	3.38
I/O/W + PCA	0.60	0.71	0.91	1.12
I/O/W + robust PCA	0.58	0.68	0.56	0.73

Table 8.1: Ground plane accuracy of the methods described in Sect. 8.2.

This problem is solved in successive methods by punishing outliers below the ground plane with the inlier/outlier/worse outlier model.

8.4.2 Drift-Corrected Odometry

We also explicitly investigate drift in height and attitude using visual odometry (VO) and drift-corrected visual odometry (DC-VO) by considering the final error in attitude and height, as can be seen in table 8.2. Even though VO alone is rather accurate already, DC-VO offers a significant improvement in both drift in height and attitude.

	VO		DC-VO	
	Δh $[\mathrm{cm}]$	$\Delta\alpha[^\circ]$	Δh $[\mathrm{cm}]$	$\Delta\alpha[^\circ]$
fr2_desk	4.58	3.96	2.45	1.66
fr3_office	7.71	1.18	0.82	0.13

Table 8.2: Final error in height and orientation.

8.5 Conclusion

In this chapter we presented a computationally efficient, accurate, and robust ground-plane detection algorithm. The main contribution is a novel inlier/outlier/worse-outlier model that can be used both within RANSAC and for robust refinement. We showed that this significantly improves the accuracy of visual odometry estimates by correcting the inevitable drift in height and attitude.

Chapter 9

A full SLAM Back-End for Parallel Tracking and Mapping

The previous chapters each focused on one aspect of a visual SLAM system: In chapter 5 we showed how to combine 2D and depth measurements, which is the foundation of this dissertation. In chapter 6 we implemented an alternative SLAM system with implicit loop closure detection. Chapter 7 is about explicit loop closure detection, and the previous chapter 8 showed how to use ground plane measurements for drift correction.

In this chapter we describe the combination of most aspects of the previous chapters within a full SLAM system. We achieve this by treating our modified version of PTAM as a SLAM front-end and implementing a suitable SLAM back-end.

9.1 PTAM as a Visual Odometry SLAM Front-End

We use our heavily modified fork of PTAM (see Klein and Murray (2007)) as a visual odometry system. Our previous modifications include porting PTAM to ROS and using depth measurements (if available) in addition to 2D measurements for map points initialization, tracking, and bundle adjustment as described in chapter 5 and in Scherer *et al.* (2012). We remove old keyframes and corresponding map points and only keep the most recent keyframes for tracking and local bundle adjustment as described in Schauwecker *et al.* (2012a). When tracking fails for a few frames, we blindly predict the pose based on the last known velocity before giving up on the current map and reinitializing it.

Limiting the number of map points per keyframe is necessary since their number and thus the time required for tracking may vary uncontrollably for different environments. We solve this problem by disregarding all but the n best map points per image pyramid level, ordered according to their Shi-Tomasi score (see Shi and Tomasi (1994)).

As mentioned above, we remove keyframes from PTAM in order to limit its computational complexity. Just deleting these keyframes forever, however, disregards possibly useful information. Instead of deleting, we now publish all keyframes with all their information before they are deleted so they can be picked up by a SLAM back-end.

9.2 SLAM Back-End

The SLAM back-end is in charge of building maps given keyframes that were deleted by the visual odometry front-end (i.e. PTAM), which includes detecting and closing loops to counter long-term drift and to keep the map consistent. Fig. 9.1 shows the general principle of our SLAM back-end working in parallel with and extending PTAM: PTAM's tracking thread is operating at camera rate, computing a pose estimate for each incoming image. Its mapping thread becomes active whenever there is a new keyframe and will optimize the relatively small local map of the few latest keyframes using bundle adjustment.

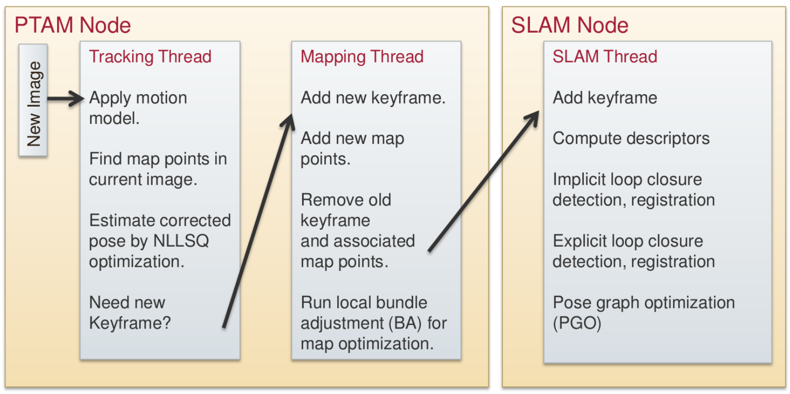

PTAM Node
New Image
Tracking Thread
Apply motion model.
Find map points in current image.
Estimate corrected pose by NLLSQ optimization.
Need new Keyframe?
Mapping Thread
Add new keyframe.
Add new map points.
Remove old keyframe and associated map points.
Run local bundle adjustment (BA) for map optimization.
SLAM Node
SLAM Thread
Add keyframe
Compute descriptors
Implicit loop closure detection, registration
Explicit loop closure detection, registration
Pose graph optimization (PGO)

Figure 9.1: Functional overview of PTAM front-end and SLAM back-end

The SLAM back-end waits for keyframes to be removed from PTAM and adds these to its own map. Whereas PTAM and the camera driver are run at camera rate as nodelets (modules with separate threads, but within the same process) to minimize overhead when transferring images, the SLAM back-end runs in its own process at a lower priority so it does not slow down tracking and only irregularly becomes active whenever a keyframe is removed from PTAM and transferred to the back-end. For each such incoming keyframe it will try to perform the following steps:

9.2.1 Keyframe Conversion

The back-end uses a keyframe representation similar to PTAM: A keyframe consists of a scale-space image pyramid with FAST corners computed on every level. What is changed is that map points are now stored within their source keyframe and that we rely on local descriptors instead of small image patches for matching.

PTAM uses a global representation for map points: All map point positions are stored in a global reference frame. This is impractical for pose graph optimization, so we instead store map points relative to their source keyframe, i.e. in which they were observed for the first time. This means that map point positions will be adjusted implicitly during pose graph optimization.

Finding map points within an image in PTAM relies on comparing warped templates and minimizing their zero-mean sum of squared differences (ZMSSD). This is feasible for tracking as in PTAM, where the image region in which a map point is searched for is small, but not for wide-baseline matching as required when closing loops. We thus compute BRIEF descriptors (see Calonder *et al.* (2010)) for all corners and map points. We chose BRIEF because it is fast and we rely on neither explicit scale invariance (which is implicitly obtained using a scale-space pyramid) nor rotation invariance (since we are using a forward-looking camera, flying close to hovering at all times) in a descriptor.

9.2.2 Retry Registration for Tracking Failures

Visual tracking in PTAM can sometimes fail and completely loose track for various reasons, which leads to rather inaccurate pose estimates that are computed solely based on the prediction obtained from applying the motion model alone until tracking can resume. If this is the case, we handle these likely inaccurate relative pose estimates by adding a so-called *motion model edge* between two affected keyframes. We also try to register these keyframes again using the descriptor-based matching and registration described in section 9.2.4. The reason behind this is that there might be short very difficult periods when the camera temporarily does not see any visual features, e.g. because its view is blocked by a textureless obstacle or because of fast motion leading to blurred images. As soon as the obstacle is out of the way or the motion has stabilized, we might be able to register the two keyframes created before and after this period.

9.2.3 Implicit and Explicit Loop Closure Detection

A trivial method of finding promising loop closure candidates is considering old keyframes whose pose estimates are so close to the current keyframe that there should be considerable overlap between the geometry visible in both views, which we call *implicit* loop closure detection. We ignore keyframes that are either too close in time to the current keyframe (since we expect drift in odometry to be negligible in this case) or geometrically too far way (since matching is unlikely to succeed). Among the remaining keyframes, we find the one with the most map points visible within the image of the current keyframe by projecting map points according to current pose estimates. If we can successfully register both keyframes (see Sect. 9.2.4), we add a loop closure edge to the pose graph.

After longer loops, too much drift might have accumulated to find loops based on geometrical closeness of keyframe pose estimates. This is why we also consider loop closure candidates based on appearance or similarity of keyframes alone instead, which we call *explicit* loop closure detection.

We use a hierarchical bag-of-words approach described in Galvez-Lopez and Tardos (2012) to retrieve the previous keyframe which is most similar to the current one and try to register this and the latest keyframe. Again, if we succeed in registering both keyframes, we add a loop closure edge.

9.2.4 Keyframe-to-Keyframe Registration

Attempts to register a pair of keyframes (A, B) start by matching map points of keyframe B to corners of A. Using these matches, we can compute a pose estimate by solving the PnP problem. We do this by applying Gao's solution to the P3P problem described in Gao *et al.* (2003) within a preemptive RANSAC scheme (see Nistér (2005)) to identify inliers and arrive at a rough estimate for the relative pose ${}^{A}T_{B}$. In a second step, we estimate the inverse relative pose ${}^{B}T_{A}$ by matching map points of A to corners of B. If there were enough inliers in both cases and both relative poses agree, we refine the relative pose using nonlinear optimization, minimizing the 2D reprojection error of all map point/corner pairs that were inliers, and insert an edge between both keyframes into the pose graph.

In the beginning, we also considered matching all corners to corners and running full bundle adjustment on all matches. This proved to be much slower than the method described above, though, for several reasons: It requires matching many more feature descriptors since there are typically more than ten times as many corners as map points, which due to its square run-time complexity leads to matching time increased by the factor of 100. Registering 2D to 2D corners requires the application of slower algorithms than P3P within a RANSAC scheme: Ideally, this would be a five-point algorithm, which requires many more model hypothesis to be created than using only 3 points. Finally, it would require optimizing both relative keyframe pose and map point positions. We do not want to add new map points or change their relative positions once they are transferred to the back-end, so this increased effort would not be of much use.

9.2.5 Pose Graph Optimization

Keyframe-Keyframe Edges

The final pose graph consists of several binary edges between keyframes that were either computed by visual odometry, by blindly applying the motion model if visual odometry failed, or by keyframe-keyframe registration within the backend. We assign the same constant weights to both visual odometry and keyframe-to-keyframe edges, and a much lower weight to motion model edges, since their relative poses are rather uncertain.

Ground Plane Edges

If we found a ground plane in the original image that was made a keyframe by PTAM, this is an incomplete absolute pose measurement we should also use in pose graph optimization. We utilize this information by adding special unary edges with the following two reprojection errors for height and attitude:

$$
\begin{aligned}
e_{att} &= \operatorname{acos}\left(\left({}^{C_i}\boldsymbol{R}_W \cdot {}^{W}\boldsymbol{up}\right)^T \cdot {}^{C_i}\boldsymbol{up}_{meas}\right) \\
e_h &= h_{meas} - h_{exp}
\end{aligned}
$$

The first element is on the angle between measured and expected up direction, the second element is the error in height. Both the expected up direction ${}^{C_i}\boldsymbol{up}$ and height origin h_{exp} can be found in the third row of the homogeneous transformation matrix that corresponds to the current pose estimate of the keyframe pose.

$$
\left({}^{W}\boldsymbol{T}_{C_i}\right)_3 = \begin{pmatrix} {}^{C_i}\boldsymbol{up}^T & h_{exp} \end{pmatrix}
$$

Implementation

The actual optimization of our pose graph is implemented using g2o (see Kümmerle *et al.* (2011)), which supports deriving custom SLAM edges and thus allows us to easily integrate different measurement types, namely relative pose edges and ground plane edges.

9.3 Experiments and Results

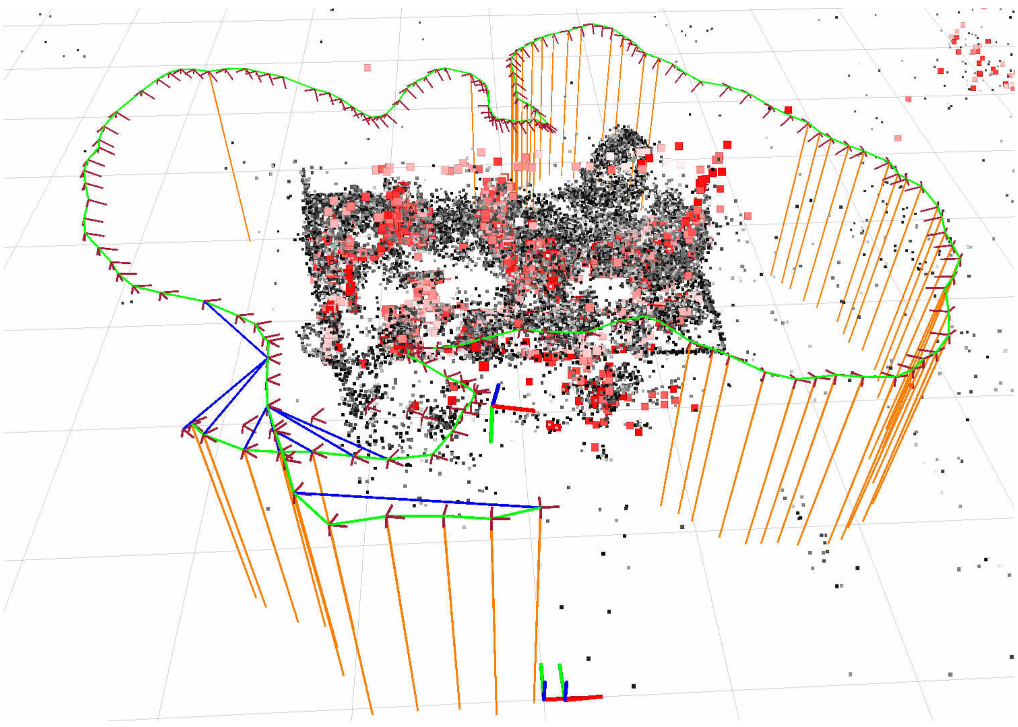
9.3.1 Accuracy Evaluation on Benchmark Dataset

We again evaluate the accuracy of this SLAM system using the TUM RGBD benchmark dataset described in Sturm *et al.* (2012). The results are shown in table 9.1.

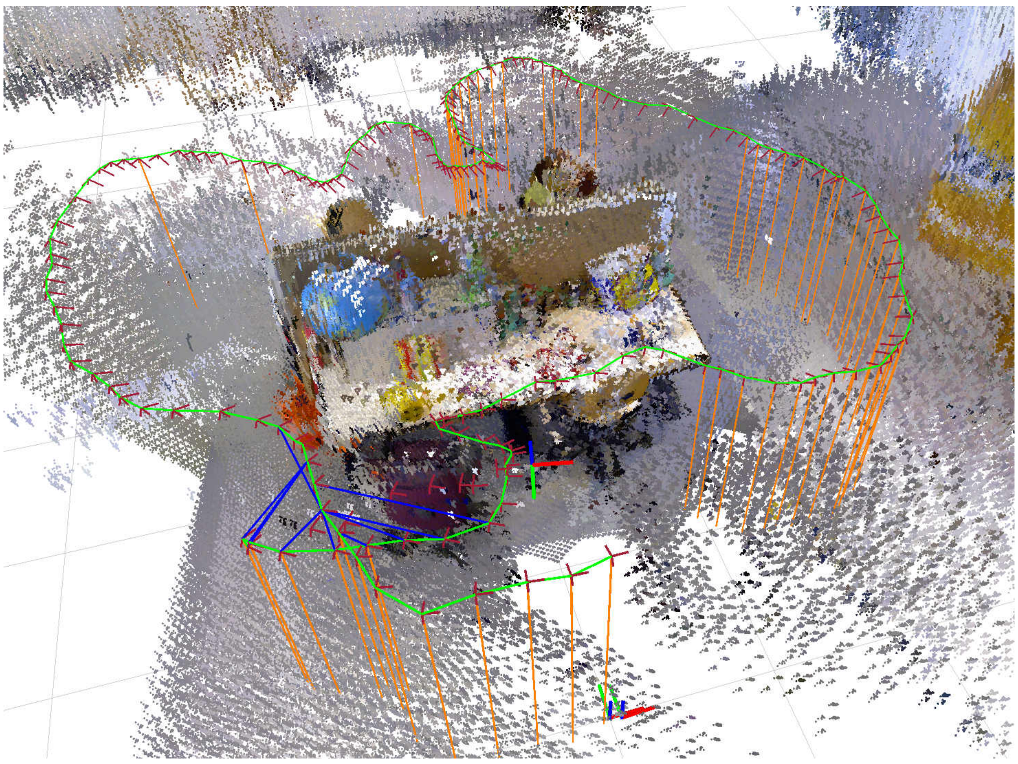
	visual odometry		drift-corrected VO		SLAM
	ATE [cm]	RPE [cm/s]	ATE [cm]	RPE [cm/s]	ATE [cm]
fr1_room	12.67	4.48	n/a	n/a	8.16
fr2_desk	7.69	1.83	7.58	1.62	8.44
fr3_office	4.18	1.17	3.88	1.46	2.10
fr3_office(v)	2.51	1.01	3.29	1.19	3.19

Table 9.1: Visual odometry accuracy on several datasets in terms of absolute tracking error (ATE) and relative position error (RPE) as determined by the RGBD benchmark tool.

We can see that the results of our SLAM system are consistently better than using visual odometry alone, which is due to the closed loops. The visualization of the result of



(a) Map built by our SLAM system



(b) Point cloud reconstructed from the map

Figure 9.2: Results from applying the proposed SLAM system to the fr3_office dataset.

our method on the fr3_office dataset is shown in figure 9.2: Keyframe poses (i.e. nodes in the pose graph) are depicted by small red coordinate frames. If there is a ground plane measurement for a keyframe, this is illustrated by the shortest line connecting the keyframe position to the ground plane drawn in orange. Visual odometry edges connecting consecutive keyframes are shown in green whereas loop closure edges are shown in blue. Figure 9.2a shows only the sparse set of map points, i.e. points that are actually used for localization. Map points currently used by PTAM in the front-end are shown in red, map points that were moved to the back-end are shown in gray. Figure 9.2b, on the other hand, shows a more dense colored point cloud reconstructed using RGBD image pairs and the keyframe poses determined by our SLAM back-end.

9.3.2 Autonomous Flight of our MAV

We first demonstrate that the proposed system enables our MAV to fly completely autonomously when following a set of predefined waypoints. We do this by commanding the MAV to fly the same rectangle three times in a row within our laboratory, all the time looking in the same absolute direction. A screenshot of the visualization containing a part of the built map can be seen in Fig. 9.3. The current pose estimate is depicted using

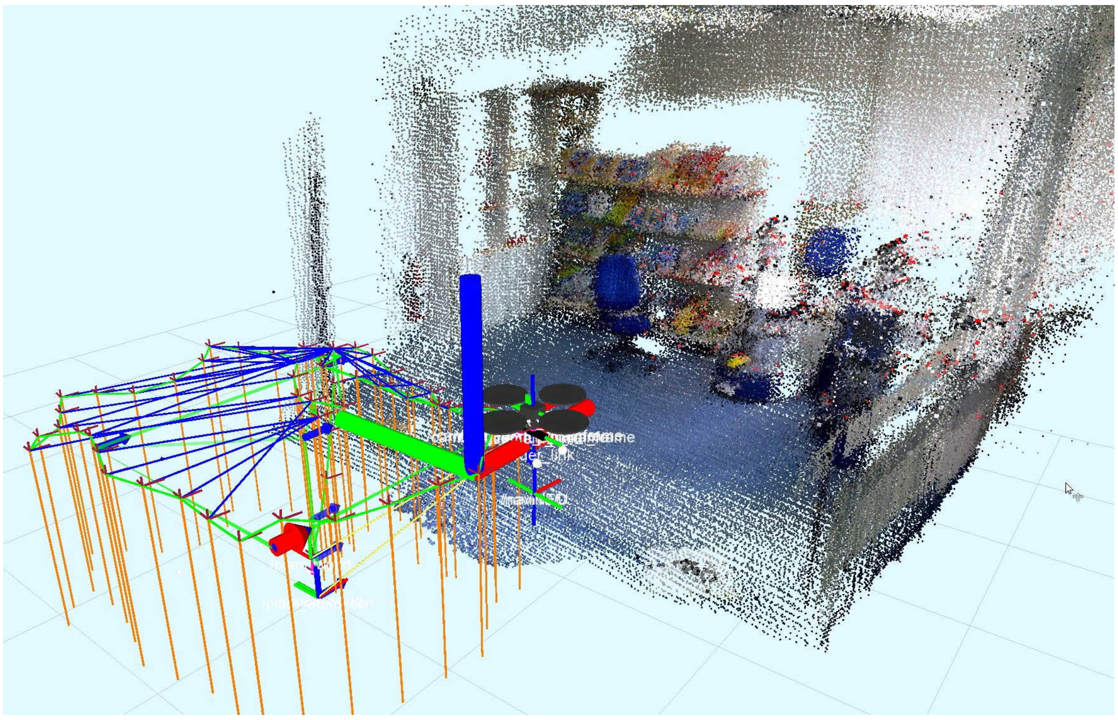


Figure 9.3: Visualization while our MAV is flying fully autonomously by repeatedly following a predefined path within our robot laboratory.

the simple quadrotor model. The red arrow is the currently active waypoint, whereas the big coordinate frame is the currently active set point for position control. We also show keyframes (small red frames), visual odometry edges (green), loop closure edges (blue) and ground plane edges (orange). Within the dense point cloud reconstruction, one can see a few very small red points, which are the map points currently used by PTAM for localization.

Even though the visual odometry estimate alone drifts over time, the large number of closed loops ensures that the path of the MAV does not drift much overall.

9.3.3 Semi-Autonomous Flight

We also show a semi-autonomous flight in order to demonstrate that we can also map slightly larger environments using on-board processing alone. We rely on semi-autonomous operation in this case, which means that the operator can modify the set point of the position controller of the MAV by pushing different buttons for moving the set point forward or backward, left or right, up or down, or rotating it left or right. A visualization of the map built by the MAV in flight and the final map of the room is shown in Fig. 9.4. We commanded the MAV to explore the room a little more such that it can better map the whole room in this case.

If one zooms into the visualization, one can see that visual odometry was rather incorrect when the MAV was turning around its axis and facing the right hand side of the room. This is due to the repetitive structure of the heating elements and different lighting conditions when directly facing the windows. The resulting map is still consistent because of successful loop closures after the MAV turned back to its original position.

9.3.4 Evaluation: Processing Time

We evaluate the processing time required for individual steps of both tracking and the SLAM back-end while our MAV is flying semi-automatically through our laboratory. The measured times are shown in Table 9.2. During this ca. 5 minutes long flight, it created a map consisting of 307 keyframes in total. Fast tracking is imperative to enable autonomous flight, especially when using only a basic PD controller. It is obvious that our combination of ground plane detection and tracking using PTAM can conveniently be computed onboard at camera rate (30 Hz), even though there are now three major threads competing for two cores of the CPU. Image preparation includes converting the original RGB image to grayscale, which is still required.

9.3.5 Application to Other Mobile Robots

Even though this work is mainly aimed at enabling aerial robots to fly autonomously, it can also directly be applied to ground mobile robots without any modifications. An example of a bigger map with more loops, reconstructed from data logged from a Mi-

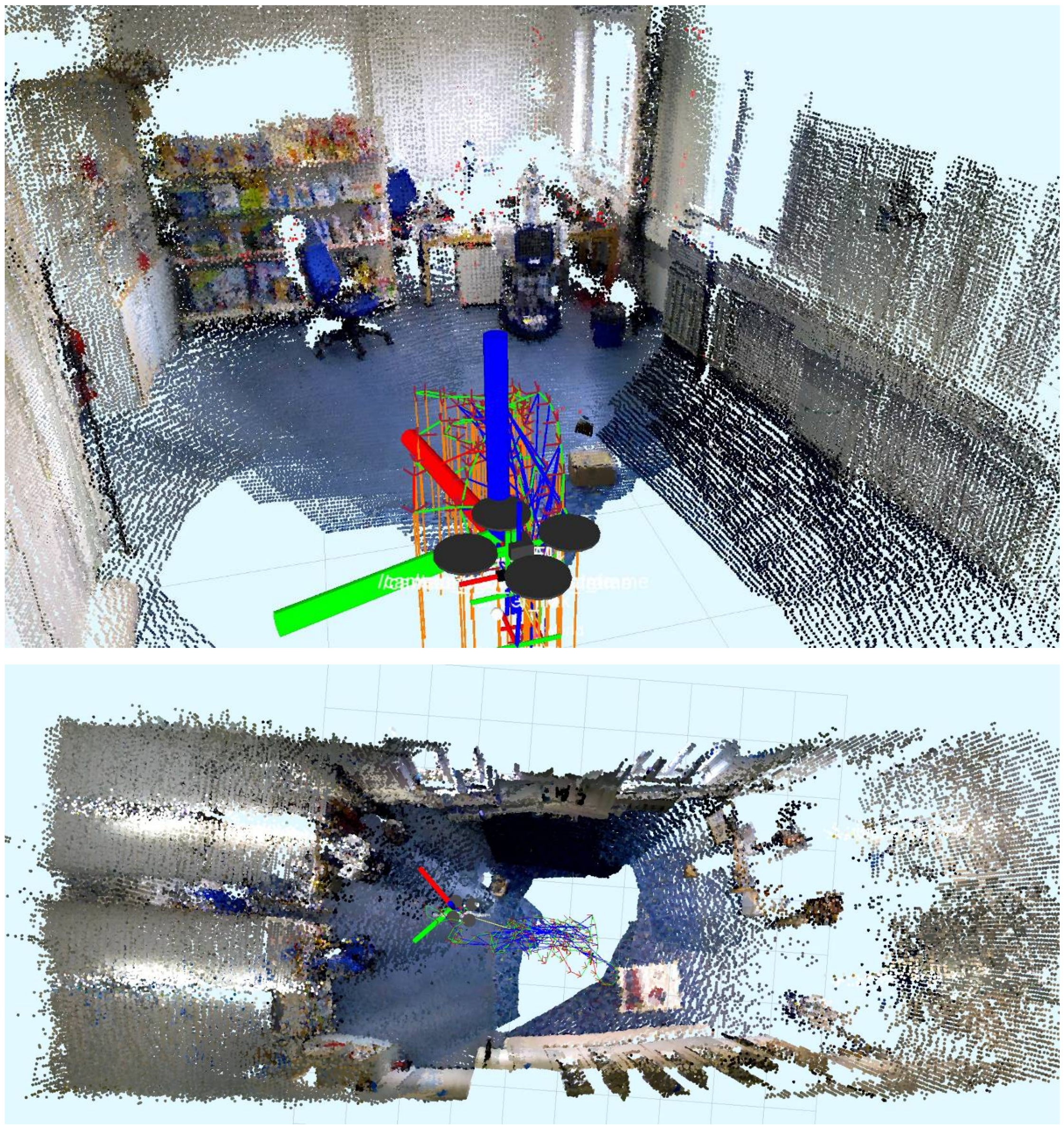


Figure 9.4: Visualization while our MAV is flying fully autonomously by following a predefined path within our robot laboratory (top) and reconstruction of the room at the end of the flight (bottom).

step	time [ms]	step	time [ms]
image prep.	1.55 ± 0.84	KF conversion	20.34 ± 8.96
ground plane	0.96 ± 0.75	loop detection	48.71 ± 27.42
tracking	22.17 ± 13.42	registration	59.68 ± 26.16
total	24.79 ± 13.85	PGO iteration	5.62 ± 4.16

Table 9.2: Computation times required by steps for tracking (left) and the SLAM back-end (right).

crosoft Kinect camera mounted on a mobile robot driving through our library is shown in Fig. 9.5.

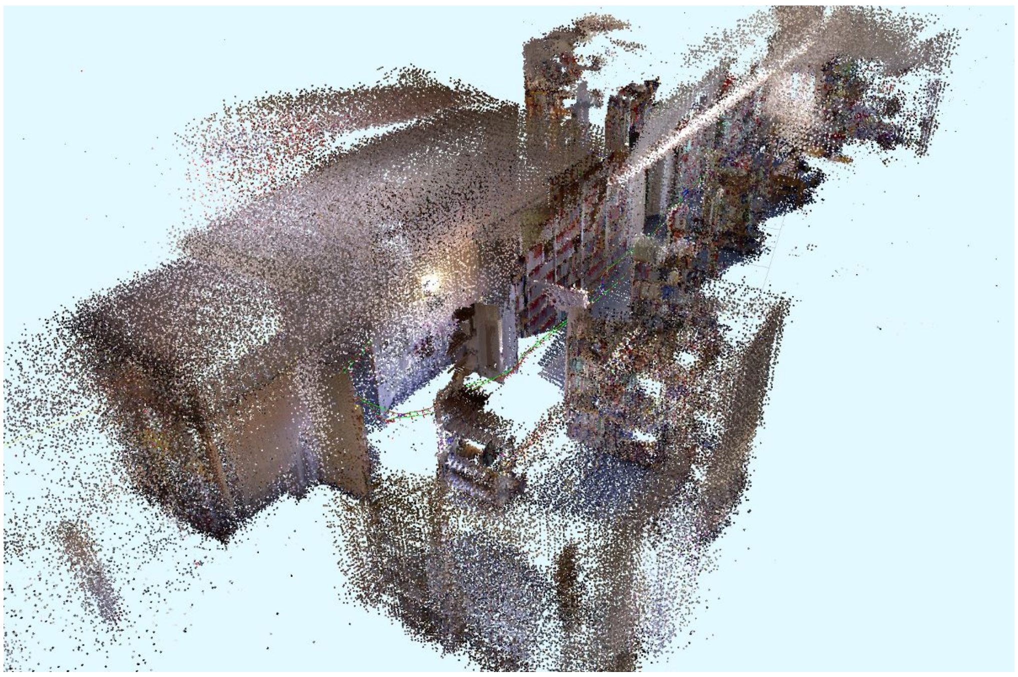


Figure 9.5: Full point cloud reconstructed from data gathered by a mobile robot driving through our library.

9.4 Conclusion

In this chapter, we developed a SLAM back-end to complement our modified PTAM version to a full visual SLAM system that integrates depth measurements whenever available as described in chapter 5, is a full SLAM system that can handle closed loops due to a SLAM back-end responsible for pose graph optimization similar to the sytem described in chapter 6, contains explicit loop closure detection using a hierarchical bag-of-words approach as investigated in chapter 7, and can integrate incomplete absolute pose measurements from ground planes detected using the method described in chapter 8. We demonstrated its accuracy and performance both on public benchmark datasets and in actual autonomous flight experiments.

Chapter 10
Conclusions

10.1 Summary

In this dissertation, we developed a full visual SLAM system that allows autonomous navigation of an MAV equipped with an RGBD camera. We observed problems with unreliable depth measurements early on in our experiments, which is reflected in the general idea behind our approach: Starting from monocular visual SLAM and additionally integrating depth measurements whenever they are available. This idea, its mathematical model, and its implementation by means of extending the popular visual SLAM system PTAM is shown in chapter 5.

Due to some major limitations of PTAM, namely the inability to detect and handle closed loops, we decided to implement an alternative SLAM system with a SLAM back-end that performs pose graph optimization instead of global bundle adjustment, which is described in chapter 6. This was to our knowledge the first SLAM system with full pose graph optimization being performed in real time on a micro aerial vehicle.

We investigated the feasibility of detecting closed loops using depth instead of intensity images in chapter 7. We found the results using depth images alone not as good but surprisingly close to a reference implementation using SIFT features on intensity images, which seemed promising for future work on real-time SLAM using depth images alone.

In order to cope with long-term drift in visual odometry, we developed a novel ground plane detection algorithm in chapter 8, which is based on a novel inlier/outlier/worse-outlier model for RANSAC and robust least squares. We showed that this method is robust, accurate, and computationally efficient and can be used to correct inevitable drift in height and attitude of visual odometry.

We finally combined the main ideas from all technical chapters in one full SLAM system as described in chapter 9: By using our modified version of PTAM with integrated depth measurements and drift-correction as a SLAM front-end and complementing it with a SLAM back-end responsible for detecting closed loops and global pose graph optimization, we managed to create a full SLAM system capable of allowing our MAV to fly fully autonomously while mapping a previously unknown environment.

While originally aimed at being used with RGBD cameras, several ideas of this dissertation could easily be transferred to applications with different cameras. The most notable examples are our method of combining 2D and depth images from chapter 5, which can directly be transferred to stereo odometry and SLAM as demonstrated in Schauwecker *et al.* (2012a), and the extension of PTAM to a full SLAM system with loop closure detection and pose graph optimization for monocular cameras as described in Yang *et al.* (2014b).

10.2 Future Work

We have successfully demonstrated that our SLAM system allows the MAV to localize itself and autonomously navigate in previously unknown environments. In all of our experiments, however, we either controlled the MAV in a semi-autonomous manner or let it follow a pre-defined path of waypoints. The next big step towards true autonomy would be robust autonomous exploration with obstacle avoidance.

The methods and algorithms required for that are theoretically available and straightforward: Keyframe-based maps produced by graph-based SLAM can be converted to an octree-based occupancy grid map (see Wurm *et al.* (2010)), which can in turn be used for planning paths through free space to frontiers that should be explored next (see Yamauchi (1997)), while avoiding obstacles. Constructing one occupancy grid map from a keyframe-based map with a large number of keyframes is computationally expensive, and the naive approach would require this after every closed loop and the following global optimization. This effect can be mitigated by applying hybrid metric-topological maps as proposed for the 2D case in Konolige *et al.* (2011) or in 3D in Schmuck *et al.* (2016).

The major hurdle preventing this so far was the availability of enough computational power at a weight low enough to allow it being carried by an MAV. Depending on the desired accuracy and the accordingly chosen parameters, the systems described in this dissertation can easily use two CPU cores to their full capacity. With more powerful onboard computers hosting more and more CPU cores becoming available, however, this is about to change.

But also the localization result of the methods described in this dissertation and their robustness might be improved. One interesting idea would be mounting multiple RGBD cameras on one MAV to mitigate the effect of the rather small viewing angle of current RGBD cameras similar to the method described in Yang *et al.* (2014c).

There are also two alternative methods that deserve to be watched closely: EKF-SLAM based visual odometry is experiencing a revival (see e.g. Li and Mourikis (2013)), since it makes some things easier, e.g. integrating inertial measurements and coping with rolling shutters. Also, dense methods Kerl *et al.* (2013a) will become more and more important with more computing power available on MAVs.

Abbreviations

BA	Bundle Adjustment
BOW	Bag Of Words
BRIEF	Binary Robust Independent Elementary Features
EKF	Extended Kalman Filter
ESC	Electric Speed Controller
ESM	Efficient Second-order Minimization
ETS	External Tracking System
GPS	Glopal Positioning System
FOV	Field of View
HOG	Histogram of Oriented Gradients
ICP	Iterative Closest Points
IMU	Inertial Measurement Unit
IRLS	Iteratively Reweighted Least Squares (estimation)
KDES	Kernel DEScriptors
LSH	Locality-Sensitive Hashing
LSQ	Least SQuares (estimation)
MAE	Mean Absolute Error
MAP	Maximum a Posteriori (estimation)
MAV	Micro Aerial Vehicle
ML	Maximum Likelihood (estimation)
NARF	Normal Aligned Radial Features
PCA	Principal Component Analysis
PGO	Pose Graph Optimization
PTAM	Parallel Tracking and Mapping
RANSAC	RANdom SAmple Consensus
RGB(D)	Red, Green, Blue (Depth)
RMSE	Root Mean Square Error
ROS	Robot Operating System
$\mathrm{SE}(3)$	Special Euclidean group in 3 dimensions, i.e. the Lie group of 3D rigid body transformations
$\mathfrak{se}(3)$	Lie algebra corresponding to $\mathrm{SE}(3)$
SfM	Structure from Motion
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping

$\mathrm{SO}(3)$	Special Orthogonal group in 3 dimensions, i.e. the group of 3D rotations
$\mathfrak{so}(3)$	Lie algebra corresponding to $\mathrm{SO}(3)$
ZMSSD	Zero-Mean Sum of Squared Differences

Bibliography

Agarwal, S., Mierle, K., and Others (2014). Ceres solver. `https://code.google.com/p/ceres-solver/`. Accessed: 2014.03.17.

Amazon.com, Inc. (2013). Amazon prime air. `http://www.amazon.com/b?ref_=tsm_1_yt_s_amzn_mx3eqp&node=8037720011`. Accessed: 02.12.2013.

Bachrach, A., Prentice, S., He, R., and Roy, N. (2011). Range–robust autonomous navigation in gps-denied environments. *Journal of Field Robotics*, **28**(5), 644–666.

Bailey, T. and Durrant-Whyte, H. (2006). Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, **13**(3), 108–117.

Baker, S. and Matthews, I. (2001). Equivalence and efficiency of image alignment algorithms. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–1090. IEEE.

Bandemann, P. (2010). Schadstoffe aufspüren - Feuerwehr setzt auf Flugroboter. `http://www.ruhrnachrichten.de/staedte/dortmund/Schadstoffe-aufspueren-Feuerwehr-setzt-auf-Flugroboter;art930,1126338` Accessed: 12.12.2014.

Barnard, S. T. and Fischler, M. A. (1982). Computational stereo. *ACM Computing Surveys (CSUR)*, **14**(4), 553–572.

Beard, R. W. and McLain, T. W. (2012). *Small unmanned aircraft: Theory and practice*. Princeton University Press.

Benhimane, S. and Malis, E. (2004). Real-time image-based tracking of planes using efficient second-order minimization. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, pages 943–948. IEEE.

Besl, P. J. (1988). Active, optical range imaging sensors. *Machine vision and applications*, **1**(2), 127–152.

Besl, P. J. and McKay, N. D. (1992). Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics.

Biber, P. and Straßer, W. (2003). The normal distributions transform: A new approach to laser scan matching. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2743–2748. IEEE.

Björck, A. (1996). *Numerical methods for least squares problems*. Siam.

Blanco, J.-L. (2010). A tutorial on se(3) transformation parameterizations and on-manifold optimization. Technical report, University of Malaga.

Bo, L., Ren, X., and Fox, D. (2010). Kernel Descriptors for Visual Recognition. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 244–252.

Bouguet, J.-Y. (2001). Pyramidal implementation of the lucas kanade feature tracker: Description of the algorithm. Technical report.

Bry, A., Bachrach, A., and Roy, N. (2012). State estimation for aggressive flight in gps-denied environments using onboard sensing. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1–8. IEEE.

Bundeswehr (2013). Mikado - das kleine auge des heeres. `http://www.deutschesheer.de/portal/a/heer/!ut/p/c4/04_SB8K8xLLM9MSSzPy8xBz9CP3I5EyrpHK9jNTUIr2S1OSMvMxsvZzStBK93MzsxJR8_YJsROUALOkjaQ!!/` Accessed: 30.11.2014.

Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). Brief: Binary robust independent elementary features. In *Computer Vision–ECCV 2010*, pages 778–792. Springer.

Calonder, M., Lepetit, V., Ozuysal, M., Trzcinski, T., Strecha, C., and Fua, P. (2012). BRIEF: Computing a Local Binary Descriptor Very Fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **34**(7), 1281–1298.

Campbell, N. (1980). Robust procedures in multivariate analysis i: Robust covariance estimation. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **29**, 231–237.

Chen, Y. and Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and vision computing*, **10**(3), 145–155.

Chum, O., Matas, J., and Obdržálek, Š. (2004). Enhancing ransac by generalized model optimization. In *Proc. of the Asian Conference on Computer Vision (ACCV)*, volume 2, pages 812–817, Seoul, Korea South. Asian Federation of Computer Vision Societies.

Csurka, G., Dance, C. R., Fan, L., Willamowski, J., and Bray, C. (2004). Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22.

Cummins, M. and Newman, P. (2008). FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance. *The International Journal of Robotics Research*, **27**(6), 647–665.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE.

Davison, A. (2003). Real-time simultaneous localisation and mapping with a single camera. In *Proc. International Conference on Computer Vision, Nice*.

Davison, A., Cid, Y. G., and Kita, N. (2004). Real-time 3D SLAM with wide-angle vision. In *Proc. IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon*.

Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **29**(6), 1052–1067.

DeMenthon, D. and Davis, L. S. (1992). Exact and approximate solutions of the perspective-three-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**(11), 1100–1105.

Devernay, F. and Faugeras, O. (2001). Straight lines have to be straight. *Machine vision and applications*, **13**(1), 14–24.

Dobler, C. (2009). *Development of Flight Hardware for a Next Generation Autonomous Micro Air Vehicle*. Master's thesis, ETH Zürich.

Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *Robotics & Automation Magazine, IEEE*, **13**(2), 99–110.

Durrant-Whyte, H., Rye, D., and Nebot, E. (1996). Localization of autonomous guided vehicles. In *Robotics Research*, pages 613–625. Springer.

Eade, E. (2008). *Monocular Simultaneous Localisation and Mapping*. Ph.D. thesis, University of Cambridge.

Eggert, D. W., Lorusso, A., and Fisher, R. B. (1997). Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Mach. Vision Appl.*, **9**, 272–290.

Engelhard, N., Endres, F., Hess, J., Sturm, J., and Burgard, W. (2011). 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*, Vasteras, Sweden.

Faugeras, O., Luong, Q.-T., and Papadopoulou, T. (2001). *The Geometry of Multiple Images: The Laws That Govern The Formation of Images of A Scene and Some of Their Applications*. MIT Press, Cambridge, MA, USA.

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

Foix, S., Alenya, G., and Torras, C. (2011). Lock-in time-of-flight (tof) cameras: a survey. *Sensors Journal, IEEE*, **11**(9), 1917–1926.

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, **1999**, 343–349.

Fraundorfer, F., Heng, L., Honegger, D., Lee, G. H., Meier, L., Tanskanen, P., and Pollefeys, M. (2012). Vision-based autonomous mapping and exploration using a quadrotor mav. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4557–4564. IEEE.

Gabay, D. (1982). Minimizing a differentiable function over a differential manifold. *Journal of Optimization Theory and Applications*, **37**(2), 177–219.

Galvez-Lopez, D. and Tardos, J. D. (2011). Real-time loop detection with bags of binary words. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 51 –58.

Galvez-Lopez, D. and Tardos, J. D. (2012). Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, **28**(5), 1188–1197.

Gao, X.-S., Hou, X.-R., Tang, J., and Cheng, H.-F. (2003). Complete solution classification for the perspective-three-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **25**(8), 930–943.

Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*.

Gokturk, S. B., Yalcin, H., and Bamji, C. (2004). A time-of-flight depth sensor-system description, issues and solutions. In *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*, pages 35–35. IEEE.

Grabe, V., Bülthoff, H. H., and Robuffo Giordano, P. (2012). On-board velocity estimation and closed-loop control of a quadrotor uav based on optical flow. In *IEEE International Conference on Robotics and Automation*.

Grabe, V., Riedel, M., Bülthoff, H., Robuffo Giordano, P., and Franchi, A. (2013). The telekyb framework for a modular and extendible ros-based quadrotor control. pages 19–25, Piscataway, NJ, USA. IEEE.

Granshaw, S. (1980). Bundle adjustment methods in engineering photogrammetry. *The Photogrammetric Record*, **10**(56), 181–207.

Griewank, A. (1989). On automatic differentiation. In *Mathematical Programming: Recent Developments and Applications*, pages 83–108, Amsterdam.

Grisetti, G., Kümmerle, R., Stachniss, C., Frese, U., and Hertzberg, C. (2010a). Hierarchical optimization on manifolds for online 2d and 3d mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA.

Grisetti, G., Kummerle, R., Stachniss, C., and Burgard, W. (2010b). A tutorial on graph-based slam. *Intelligent Transportation Systems Magazine, IEEE*, **2**(4), 31–43.

Grzonka, S., Grisetti, G., and Burgard, W. (2009). Towards a navigation system for autonomous indoor flying. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2878–2883. IEEE.

Hall, B. (2003). *Lie groups, Lie algebras, and representations: an elementary introduction*, volume 222. Springer.

Hannah, M. J. (1974). *Computer Matching of Areas in Stereo Images.* Ph.D. thesis, Stanford, CA, USA. AAI7427032.

Harris, C. and Stephens, M. J. (1988). A combined corner and edge detector. In *Proc. of Fourth Alvey Vision Conference*, pages 147–151.

Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition.

Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2010). RGB-D mapping: Using depth cameras for dense 3d modeling of indoor environments. In *the 12th International Symposium on Experimental Robotics (ISER)*, volume 20, pages 22–25.

Herbst, R. (2010). Unbemannte Flugobjekte „Spione“ für eine nachhaltige Landwirtschaft. `https://www.dbu.de/123artikel29597_335.html` Accessed: 29.12.2014.

Hern, A. (2014). DHL launches first commercial drone 'parcelcopter' delivery service. `http://www.theguardian.com/technology/2014/sep/25/german-dhl-launches-first-commercial-drone-delivery-service`. Accessed: 27.12.2014.

Hertzberg, C. (2008). *A Framework for Sparse, Non-Linear Least Squares Problems on Manifolds*. Master's thesis, Universität Bremen.

Hertzberg, C., Wagner, R., Frese, U., and Schröder, L. (2013). Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *Information Fusion*, **14**(1), 57–77.

Honig, Z. (2011). T-Hawk UAV enters Fukushima danger zone, returns with video. `http://www.engadget.com/2011/04/21/t-hawk-uav-enters-fukushima-danger-zone-returns-with-video/` Accessed: 28.12.2014.

Horn, B. K. (1987). Closed-form solution of absolute orientation using unit quaternions. *JOSA A*, **4**(4), 629–642.

Huang, A. S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D., and Roy, N. (2011). Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Int. Symposium on Robotics Research (ISRR),(Flagstaff, Arizona, USA)*.

Huber, P. J. (1964). Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, **35**(1), 73–101.

Huber, P. J. (1972). The 1972 wald lecture robust statistics: A review. *The Annals of Mathematical Statistics*, pages 1041–1067.

Huhle, B., Fleck, S., and Schilling, A. (2007). Integrating 3d time-of-flight camera data and high resolution images for 3dtv applications. In *3DTV Conference, 2007*, pages 1–4. IEEE.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, **82**(1), 35–45.

Kerl, C., Sturm, J., and Cremers, D. (2013a). Dense visual slam for rgb-d cameras. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 2100–2106. IEEE.

Kerl, C., Sturm, J., and Cremers, D. (2013b). Robust odometry estimation for rgb-d cameras. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3748–3754. IEEE.

Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan.

Kloss, A. (2012). *Wiedererkennung bekannter Orte durch Tiefenbilder*. Bachelor's thesis, University of Tuebingen, WSI-KS.

Konolige, K. and Agrawal, M. (2008). FrameSLAM: From Bundle Adjustment to Real-Time Visual Mapping. *IEEE Trans. Robotics*, **24**(5), 1066–1077.

Konolige, K., Marder-Eppstein, E., and Marthi, B. (2011). Navigation in hybrid metric-topological maps. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3041–3047. IEEE.

Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). g2o: A general framework for graph optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Shanghai, China.

Lang, M. (2013). Drohnen-Auslieferung: DHL führt Paketkopter vor. `http://www.heise.de/newsticker/meldung/Drohnen-Auslieferung-DHL-fuehrt-Paketkopter-vor-2063059.html`. Accessed: 27.12.2014.

Lange, R. (2000). 3d time-of-flight distance measurement with custom solid-state image sensors in cmos/ccd-technology.

Lee, J. M. (2003). *Smooth manifolds*. Springer.

Leonard, J. J. and Durrant-Whyte, H. F. (1991). Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, pages 1442–1447. Ieee.

Levenberg, K. (1944). A method for the solution of certain problems in least squares. *Quarterly of applied mathematics*, **2**, 164–168.

Li, M. and Mourikis, A. I. (2013). High-precision, consistent ekf-based visual–inertial odometry. *The International Journal of Robotics Research*, **32**(6), 690–711.

Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, **60**(2), 91–110.

Lu, F. and Milios, E. (1997). Globally consistent range scan alignment for environment mapping. *Autonomous robots*, **4**(4), 333–349.

Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th international joint conference on Artificial intelligence*, pages 674–679.

Ma, Y., Soatto, S., Kosecka, J., and Sastry, S. S. (2003). *An Invitation to 3-D Vision: From Images to Geometric Models*. SpringerVerlag.

Madsen, K., Nielsen, H. B., and Tingleff, O. (2004). Methods for non-linear least squares problems (2nd ed.).

Mahony, R., Kumar, V., and Corke, P. (2012). Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *Robotics & Automation Magazine, IEEE*, **19**(3), 20–32.

Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, **11**(2), 431–441.

Masselli, A., Hanten, R., and Zell, A. (2014). Localization of unmanned aerial vehicles using terrain classification from aerial images. In *2014 International Conference on Intelligent Autonomous Systems (IAS-13)*, Padova, Italy.

Meier, L. (2009). Mavlink micro air vehicle communication protocol. `http://qgroundcontrol.org/mavlink/start` Accessed: 10.12.2014.

Meier, L., Tanskanen, P., Fraundorfer, F., and Pollefeys, M. (2011). PIXHAWK: A system for autonomous flight using onboard computer vision. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2992–2997.

Molton, N. D., Davison, A. J., and Reid, I. D. (2004). Locally planar patch features for real-time structure from motion. In *Proc. British Machine Vision Conference*. BMVC. (To appear).

Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B., *et al.* (2002). Fastslam: A factored solution to the simultaneous localization and mapping problem. In *AAAI/IAAI*, pages 593–598.

Montgomery, D. C., Peck, E. A., and Vining, G. G. (2012). *Introduction to linear regression analysis*, volume 821. John Wiley & Sons.

Montiel, J., Civera, J., and Davison, A. J. (2006). Unified inverse depth parametrization for monocular slam. *analysis*, **9**, 1.

Mutto, C. D., Zanuttigh, P., and Cortelazzo, G. M. (2012). *Time-of-Flight Cameras and Microsoft Kinect(TM)*. Springer Publishing Company, Incorporated.

Nistér, D. (2005). Preemptive RANSAC for live structure and motion estimation. *Machine Vision and Applications*, **16**(5), 321–329.

Nister, D. and Stewenius, H. (2006). Scalable Recognition with a Vocabulary Tree. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2161–2168.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization, Second Edition*. Springer, New York, 2nd edition.

Olson, E. B. (2009). Real-time correlative scan matching. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4387–4393. IEEE.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5.

Rauscher, G., Dube, D., and Zell, A. (2014). A comparison of 3d sensors for wheeled mobile robots. In *2014 International Conference on Intelligent Autonomous Systems (IAS-13)*, Padova, Italy.

Reichinger, A. (2011). `https://azttm.wordpress.com/2011/04/03/kinect-pattern-uncovered/` Accessed: 24.3.2014.

Renner, E. (1995). *Pinhole photography: rediscovering a historic technique*. Focal Press.

Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer.

Rushe, D. (2014). Google reveals home delivery drone program project wing. *The Guardian*. `http://www.theguardian.com/technology/2014/aug/29/google-joins-amazon-in-testing-home-delivery-drones`. Accessed: 27.12.2014.

Samuel, H. (2014). France's la poste develops drone to deliver parcels. *The Telegraph*. Accessed: 27.12.2104.

Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, **47**(1-3), 7–42.

Scharstein, D. and Szeliski, R. (2003). High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–195. IEEE.

Schauwecker, K. and Zell, A. (2014). On-Board Dual-Stereo-Vision for the Navigation of an Autonomous MAV. *Journal of Intelligent & Robotic Systems*, **74**(1-2), 1–16.

Schauwecker, K., Ke, N. R., Scherer, S. A., and Zell, A. (2012a). Markerless Visual Control of a Quad-Rotor Micro Aerial Vehicle by Means of On-Board Stereo Processing. In *22nd Conference on Autonomous Mobile Systems (AMS)*, pages 11–20, Stuttgart, Germany. Springer.

Schauwecker, K., Klette, R., and Zell, A. (2012b). A new feature detector and stereo matching method for accurate high-performance sparse stereo matching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5171–5176, Vilamoura, Algarve, Portugal. IEEE.

Scherer, S. A. and Zell, A. (2013). Efficient Onboard RGBD-SLAM for Fully Autonomous MAVs. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1062–1068, Tokyo Big Sight, Japan.

Scherer, S. A., Dube, D., Komma, P., Masselli, A., and Zell, A. (2011). Robust Real-Time Number Sign Detection on a Mobile Outdoor Robot. In *Proceedings of the 6th European Conference on Mobile Robots (ECMR 2011)*, Örebro, Sweden.

Scherer, S. A., Dube, D., and Zell, A. (2012). Using depth in visual simultaneous localisation and mapping. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 5216–5221, St. Paul, Minnesota, USA.

Scherer, S. A., Kloss, A., and Zell, A. (2013). Loop Closure Detection using Depth Images. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 100 – 106, Barcelona, Catalonia, Spain.

Scherer, S. A., Yang, S., and Zell, A. (2015). DCTAM: Drift-corrected tracking and mapping for autonomous micro aerial vehicles. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 1094–1101, Denver, CO, USA.

Schmuck, P., Scherer, S. A., and Zell, A. (2016). Hybrid metric-topological 3d occupancy grid maps for large-scale mapping. *IFAC-PapersOnLine*, **49**(15), 230 – 235.

Schällibaum AG (2014). vAIRmessung - Drohnenvermessung. `http://www.schaellibaum.ch/geomatik/vairmessung-drohnenvermessung/`. Accessed 27.12.2014.

Segal, A., Haehnel, D., and Thrun, S. (2009). Generalized-icp. In *Robotics: Science and Systems*, volume 2.

Shahbazi, H. and Zhang, H. (2011). Application of Locality Sensitive Hashing to real-time loop closure detection. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1228–1233.

Shen, S., Michael, N., and Kumar, V. (2011). Autonomous multi-floor indoor navigation with a computationally constrained mav. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 20–25. IEEE.

Shi, J. and Tomasi, C. (1994). Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 593–600.

Sibley, G., Mei, C., Reid, I., and Newman, P. (2009). Adaptive relative bundle adjustment. In *Robotics Science and Systems Conference*, pages 1–8.

Sibley, G., Mei, C., Reid, I., and Newman, P. (2010). Vast-scale outdoor navigation using adaptive relative bundle adjustment. *The International Journal of Robotics Research*, **29**(8), 958–980.

Sivic, J. and Zisserman, A. (2003). Video google: A text retrieval approach to object matching in videos. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1470–1477. IEEE.

Slama, C. C., Theurer, C., and Henriksen, S. W., editors (1980). *Manual of Photogrammetry*. American Society of Photogrammetry.

Smith, R., Self, M., and Cheeseman, P. (1990). Estimating uncertain spatial relationships in robotics. In *Autonomous robot vehicles*, pages 167–193. Springer.

Smith, R. C. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, **5**(4), 56–68.

Smith, S. T. (1993). *Geometric Optimization Methods for Adaptive Filtering*. Ph.D. thesis, Cambridge, MA, USA. UMI Order No. GAX93-31032.

Snavely, N., Seitz, S. M., and Szeliski, R. (2006). Photo tourism: exploring photo collections in 3d. *ACM transactions on graphics (TOG)*, **25**(3), 835–846.

Steder, B., Rusu, R., Konolige, K., and Burgard, W. (2011). Point feature extraction on 3D range scans taking into account object boundaries. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2601–2608.

Steinbrucker, F., Sturm, J., and Cremers, D. (2011). Real-time visual odometry from dense rgb-d images. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 719–722. IEEE.

Strasdat, H., Montiel, J. M. M., and Davison, A. J. (2010). Real-time monocular slam: Why filter? In *ICRA*, pages 2657–2664. IEEE.

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

Sünderhauf, N. and Protzel, P. (2012). Towards a robust back-end for pose graph SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1254–1261. IEEE.

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition.

Tanimoto, S. and Pavlidis, T. (1975). A hierarchical data structure for picture processing. *Computer Graphics and Image Processing*, **4**(2), 104 – 119.

Taylor, C. J., Taylor, C. J., Kriegman, D. J., and Kriegman, D. J. (1994). Minimization on the lie group so(3) and related manifolds. Technical report.

Taylor, J. (1997). *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements*. A series of books in physics. University Science Books.

Thacker, N. and Lacey, A. (1996). Tutorial: The likelihood interpretation of the kalman filter. *TINA Memos: Advanced Applied Statistics*, **2**.

Thrun, S. (2003). Learning occupancy grid maps with forward sensor models. *Autonomous robots*, **15**(2), 111–127.

Thrun, S. and Leonard, J. J. (2008). *Springer Handbook of Robotics*, chapter Simultaneous Localization and Mapping, pages 871–889. Springer.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (2000). Bundle adjustment – a modern synthesis. In B. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 298–372. Springer-Verlag.

Trucco, E. and Verri, A. (1998). *Introductory Techniques for 3-D Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Weingarten, J. W., Gruener, G., and Siegwart, R. (2004). Probabilistic plane fitting in 3d and an application to robotic mapping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, pages 927–932.

Weiss, S., Scaramuzza, D., and Siegwart, R. (2011). Monocular-slam–based navigation for autonomous micro helicopters in gps-denied environments. *Journal of Field Robotics*, **28**(6), 854–874.

Weiss, S., Achtelik, M. W., Lynen, S., Chli, M., and Siegwart, R. (2012). Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 957–964. IEEE.

Wenzel, K. E. and Zell, A. (2009). Automatic Take Off, Hovering and Landing Control for Miniature Helicopters with Low-Cost Onboard Hardware. In *Autonome Mobile Systeme 2009*, pages 73–80, Karlsruhe, Germany. KIT.

Whelan, T., Johannsson, H., Kaess, M., Leonard, J. J., and McDonald, J. (2013). Robust real-time visual odometry for dense rgb-d mapping. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5724–5731. IEEE.

Wurm, K. M., Hornung, A., Bennewitz, M., Stachniss, C., and Burgard, W. (2010). Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, volume 2.

Yamauchi, B. (1997). A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146–151. IEEE.

Yang, S., Scherer, S. A., and Zell, A. (2012). An Onboard Monocular Vision System for Autonomous Takeoff, Hovering and Landing of a Micro Aerial Vehicle. In *2012 International Conference on Unmanned Aircraft Systems (ICUAS'12)*, Philadelphia, PA, USA.

Yang, S., Scherer, S. A., and Zell, A. (2013a). An Onboard Monocular Vision System for Autonomous Takeoff, Hovering and Landing of a Micro Aerial Vehicle. *Journal of Intelligent & Robotic Systems*, **69**(1–4), 499–515.

Yang, S., Scherer, S. A., Schauwecker, K., and Zell, A. (2013b). Onboard Monocular Vision for Landing of an MAV on a Landing Site Specified by a Single Reference Image. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS'13)*, pages 317–324, Atlanta, GA, USA.

Yang, S., Scherer, S. A., Schauwecker, K., and Zell, A. (2014a). Autonomous Landing of MAVs on Arbitrarily Textured Landing Sites using Onboard Monocular Vision. *Journal of Intelligent & Robotic Systems*, **74**(1-2), 27–43.

Yang, S., Scherer, S. A., and Zell, A. (2014b). Robust onboard visual SLAM for autonomous MAVs. In *2014 International Conference on Intelligent Autonomous Systems (IAS-13)*, Padova, Italy.

Yang, S., Scherer, S. A., and Zell, A. (2014c). Visual SLAM for Autonomous MAVs with Dual Cameras. In *2014 International Conference on Robotics and Automation (ICRA'14)*, Hongkong, China.

Zalevsky, Z., Shpunt, A., Maizels, A., and Garcia, J. (2010). Method and system for object reconstruction. US Patent App. 11/991,994.

Zhang, H. (2011). BoRF: Loop-closure detection with scale invariant visual features. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3125–3130.

Zhang, Q. and Pless, R. (2004). Extrinsic calibration of a camera and laser range finder (improves camera calibration). In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2301–2306. IEEE.

Zhang, Z., Zhang, Z., Robotique, P., and Robotvis, P. (1997). Parameter estimation techniques: A tutorial with application to conic fitting. *Image and Vision Computing*, **15**, 59–76.