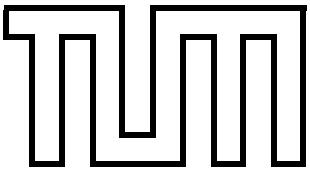


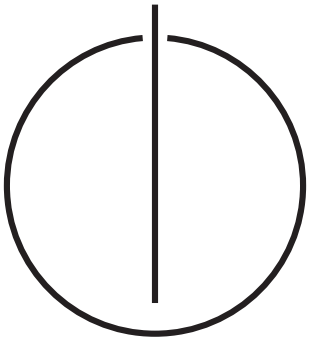
FAKULTÄT FÜR INFORMATIK

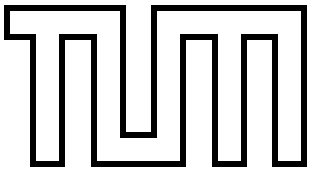
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Autonomous Camera-Based Navigation of a Quadrocopter

Jakob Julian Engel





FAKULTÄT FÜR INFORMATIK

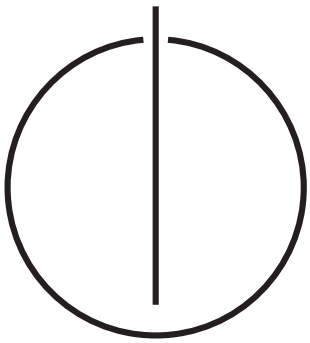
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Autonomous Camera-Based Navigation of a Quadrocopter

Autonome kamerabasierte Navigation eines Quadrocopters

Author:	Jakob Julian Engel
Supervisor:	Prof. Dr. Daniel Cremers
Advisor:	Dr. Jürgen Sturm
Date:	December 15, 2011



Ich versichere, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

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

München, den 15. Dezember 2011
Munich, December 15, 2011 Jakob Julian Engel

Contents

Abstract

In this thesis, we developed a system that enables a quadrocopter to localize and navigate autonomously in previously unknown and GPS-denied environments. Our approach uses a monocular camera onboard the quadrocopter and does not require artificial markers or external sensors.

Our approach consists of three main components. First, we use a monocular, keyframe-based simultaneous localization and mapping (SLAM) system for pose estimation. Second, we employ an extended Kalman filter, which includes a full model of the drone's flight and control dynamics to fuse and synchronize all available data and to compensate for delays arising from the communication process and the computations required. Third, we use a PID controller to control the position and orientation of the drone.

We propose a novel method to estimate the absolute scale of the generated visual map from inertial and altitude measurements, which is based on a statistical formulation of the problem. Following a maximum likelihood (ML) approach, we derive a closed-form solution for the ML estimator of the scale.

We implemented our approach on a real robot and extensively tested and evaluated it in different real-world environments. As platform we use the Parrot AR.Drone, demonstrating what can be achieved with modern, low-cost and commercially available hardware platforms as tool for robotics research. In our approach, all computations are performed on a ground station, which is connected to the drone via wireless LAN.

The results demonstrate the system's robustness and ability to accurately navigate in unknown environments. In particular we demonstrate that our system is (1) robust to temporary loss of visual tracking due to incorporation of inertial and altitude measurements and (2) is able to eliminate the odometry drift due to the incorporation of a visual SLAM system.

1. Introduction

In recent years, both remote controlled and autonomously flying Miniature Aerial Vehicles (MAVs) have become an important tool not only in the military domain, but also in civilian environments. Particularly quadcopters are becoming more popular, especially for observational and exploration purposes in indoor and outdoor environments, but also for data collection, object manipulation or simply as high-tech toys.

There are numerous example where MAVs are successfully used in practice, for example for exploratory tasks such as inspecting the damaged nuclear reactors in Fukushima in March 2011 and for aerial based observation and monitoring of potentially dangerous situations, such as protests or large scale sport events.

There are however many more potential applications: A swarm of small, light and cheap quadcopters could for example be deployed to quickly and without risking human lives explore collapsed buildings to find survivors. Equipped with high-resolution cameras, MAVs could also be used as flying photographers, providing aerial based videos of sport events or simply taking holiday photos from a whole new perspective.

Having a flying behavior similar to a traditional helicopter, a quadrocopter is able to land and start vertically, stay perfectly still in the air and move in any given direction at any time, without having to turn first. This enables quadrocopters - in contrary to traditional airplanes - to maneuver in extremely constrained indoor spaces such as corridors or offices, and makes them ideally suited for stationary observation or exploration in obstacle-dense or indoor environments.

While the concept of an aircraft flying with four horizontally aligned rotors had already been proposed in 1922 [42], this design quickly disappeared and was dominated by the much more common two-rotor helicopter. There are two main reasons for this development: While mechanically very simple, a quadrocopter is inherently unstable and hence difficult to control - without the help of advanced electronic control systems and stabilizing routines, manual control turned out to be too complex. Furthermore, quadcopters are less energy-efficient than traditional helicopters.

With the growing importance of MAVs however, the quadrocopter design has become more popular again. It is mechanically much simpler than a normal helicopter as all four rotors have a fixed pitch. Furthermore, the four rotors can be enclosed by a frame, protecting them in collisions and permitting safe flights indoors and in obstacle-dense environments. Finally, the use of four rotors allows each to have a smaller diameter, causing them to store less kinetic energy during flight and reducing the damage caused should the rotor hit an object, making quadrocopters significantly safer to use close to people.

In order to navigate, modern MAVs can rely on a wide range of sensors. In addition to an inertial measurement unit (IMU) measuring the aircraft's orientation and acceleration,

Figure 1.1. Top: the de Bothezat helicopter, built in 1922 by the US Army. It is considered to be one of the first successful helicopters ever built, however it only ever reached a hight of approximately 9 m, and stayed in the air no more than 2:45 minutes. Middle: the Hummingbird miniature quadrocopter produced by Ascending Technologies. It is used by the University of Pennsylvania's GRASP Lab [26, 25] (bottom), and is not only capable of extreme stunts such as triple-flips or flying through extremely narrow gaps, but can also be used for object manipulation by attaching a gripper to the bottom of the drone. It however relies on an advanced external tracking system, using 20 high-speed cameras distributed around the room to estimate its current position.

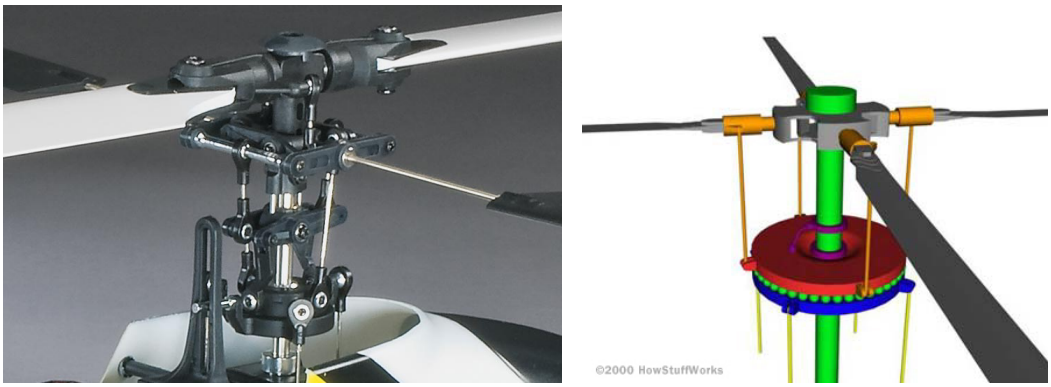


Figure 1.2. The swash plate assembly required by a traditional helicopter to steer. The orange control rods can be used to statically adjust the rotor blade's pitch, enabling the helicopter to rise and fall, while the yellow control rods allow for the swash plates to be tilted, causing the rotor blade's pitch to change during the course of each rotation and inducing a sideway movement. As a quadrocopter can be navigated simply by adjusting the rotational speed of each rotor, there is no need for such delicate mechanics.

a GPS based navigation system is often used to determine the MAV's absolute position - enabling it to autonomously navigate in a known environment, or simply to stay at its position without drifting away. Even when being controlled remotely by an experienced pilot, such systems can be of significant help, greatly reducing the pilot's workload and providing a much simpler navigational interface.

When flying in an unknown environment however - or indoors where there is no GPS signal available - alternative localization methods are required. There is a wide range of sensors available that can be used to accomplish this task: from relatively simple, small and cheap ultrasonic range sensors measuring the distance to the closest object in one particular direction, up to high-resolution laser range scanners providing a full depth image, but coming at a price of several thousand euro. One of the most direct ways to collect information about the surrounding environment are optical cameras: While providing a huge amount of information, they are comparatively cheap, energy-efficient, small and light and therefore particularly suited for MAVs. Dealing with this huge amount of visual data however has proven to be a very challenging task requiring large amounts of computational capacity.

The probably most important shortcoming of an optical camera and the reason for many of the faced challenges is the lack of depth information - as only a two-dimensional projection of the visible field is observed, distance and size of objects in the image cannot be determined.

The first and major component of any system capable of autonomous navigation is the ability to localize itself in space. This is particularly the case for an aerial vehicle - while for ground-based vehicles not moving typically is a trivial task, this is not the case for a flying robot. Holding a position in the air requires constantly counteracting minor randomly induced movements, which in turn requires a method to detect these movements. While up to a certain degree - especially with modern, high-accuracy IMUs - this is possi-

ble without taking external reference points into consideration, over time the lack of such external references will lead to small errors accumulating and a slow drift away from the desired position. With the possibility to estimate one's own position with respect to some fixed point however, this drift can be compensated for.

The task of accurately tracking the motion of a robot in an arbitrary, previously unseen environment has been the focus of a lot of research in the field of computer vision and robotics, and is widely known as the simultaneous localization and mapping (SLAM) problem. The general idea is very straight-forward: Using available sensor data, a map of the environment is generated. This map in turn is used to re-estimate the new position of the robot after a short period of time. A SLAM system thus aims at answering the two questions "What does the world look like?" and "Where am I?". This process can furthermore be done actively, that is navigating a robot such that new information about the environment can be acquired while assuring that the current pose can still be tracked accurately. Such approaches are also called SPLAM (simultaneous planning, localization and mapping) [35].

Once the MAV's position can be estimated, it can be used to approach and hold a given target position or follow a fixed path. Furthermore, such a system can be used to significantly reduce a pilot's workload, making manual control of the MAV much easier by automatically compensating for the inherent instability of the aircraft and in particular horizontal drift. In order to cope with previously unseen environments and allow for truly autonomous behavior however, knowing the MAV's position is not sufficient - one also needs ways to detect obstacles, walls, and maybe objects of interest. While the map built for SLAM can partly be used to infer information regarding the surrounding environment such as the position of obstacles, in general additional methods are required.

1.1. Problem Statement

The objective of this thesis is to develop a system capable of controlling and navigating the Parrot AR.Drone in an unknown environment using only onboard sensors, without markers or calibration objects. The main sensor to be used is the frontal camera in order to compute an absolute estimate of the drone's pose by applying visual tracking methods. This pose estimate can then be used to calculate the control commands required to fly to and hold a desired position in three-dimensional space. This approach enables a quadrocopter to accomplish tasks such as

- holding a flying position in spite of external disturbances and interference such as wind,
- high-level manual control of the drone: Instead of directly piloting the drone, this system enables the pilot to send high-level control commands such as "*move by* $(\delta x, \delta y, \delta z)^T$ *meters*" or "*fly to position* $(x, y, z)^T$",
- following a given path consisting of way points, specified in three-dimensional coordinates relative to the starting point.

Particular challenges include estimating the unknown scale of the visual map, compensating for the large delays present in the system, and dealing with the limited sensor quality available by combining visual pose estimates with additional sensor measurements available. Furthermore the system is required to be robust with respect to temporary loss of visual tracking, missing or corrupted sensor measurements and varying connection quality of the wireless link.

1.2. Outline

In **Chapter 2**, we introduce the quadrocopter used (the Parrot AR.Drone) and state its capabilities and available sensors. Being a commercial product, it offers only very limited access to hardware and onboard software - we therefore treat the drone itself as a black box, briefly summarizing the communication interface and the SDK provided by Parrot.

The following three chapters explain and detail techniques and methods used in our approach: In **Chapter 3**, techniques and mathematical methods used in state-of-the-art SLAM algorithms, in particular the components of the parallel tracking and mapping (PTAM) system by Georg Klein and David Murray [14] are presented. In **Chapter 4**, we present the data fusion and prediction method used in our system, the well-known and widely used extended Kalman filter (EKF). In **Chapter 5**, we introduce proportional-integral-differential (PID) control, a method widely used in industrial applications and in our approach for controlling the drone.

In **Chapter 6**, we propose a novel method to estimate the absolute scale of the generated visual map from inertial and altitude measurements, which is based on a statistical formulation of the problem. Following a maximum likelihood (ML) approach, we derive a closed-form solution for the ML estimator of the scale.

In **Chapter 7**, we present our approach, the developed system and its three main components: a monocular, keyframe-based SLAM system based on PTAM, an EKF for state estimation, data fusion and state prediction and the PID controller controlling the drone.

In **Chapter 8**, we evaluate the performance and accuracy of the developed system on experimental data obtained from a large number of test flights with the AR.Drone. In particular we measure the accuracy of the proposed scale estimation method, demonstrate how a SLAM system eliminates global drift and show how robustness to temporary loss of visual tracking is compensated for by incorporating IMU measurements.

Finally, in **Chapter 9** we summarize the result of this thesis, and the capabilities of the developed system. In the last chapter, **Chapter 10** we propose ways to improve and extend the current system and give an outlook on future research.

2. Quadrocopter

The Parrot AR.Drone was introduced in January 2010, originally designed as a sophisticated toy for augmented reality games. It is meant to be controlled by a pilot using a smart phone or a tablet PC. In spite of the original design as a high-tech toy, the drone quickly caught attention of universities and research institutions, and today is used in several research projects in the fields of Robotics, Artificial Intelligence and Computer Vision [17, 4]: in contrast to many other available remote controlled aerial vehicles, the drone with a retail price of only 300 € is inexpensive, robust, and easy to use and fly. In this chapter, the drone hardware and the available sensors, as well as the available software is presented: in Section 2.1, we first state the basic flight properties of a quadrocopter, and then describe the Parrot AR.Drone and the available sensors in particular. In Section 2.2, we briefly present the available software and communication interfaces for the Parrot AR.Drone.

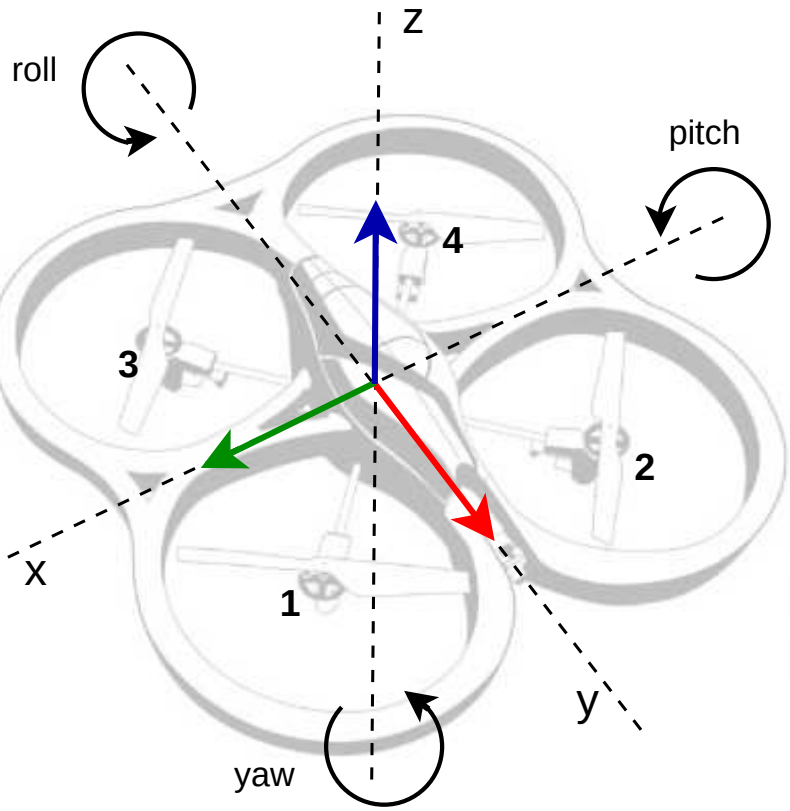


Figure 2.1. Schematics of the Parrot AR.Drone, as well as the main coordinate frame. It is common to specify the orientation (attitude) of an aircraft using roll-, pitch- and yaw-angles, defining rotation around the y, x and z axis respectively.

2.1. Hardware

2.1.1. Basic Quadrocopter Mechanics

A quadrocopter is a helicopter, which is lifted and maneuvered by four rotors. It can be maneuvered in three-dimensional space solely by adjusting the individual engine speeds (see Figure 2.1): while all four rotors contribute to the upwards thrust, two opposite ones are rotating clockwise (rotors 2 and 3) while the other two (rotors 1 and 4) are rotating counter-clockwise, canceling out their respective torques. Ignoring mechanical inaccuracies and external influences, running all engines at equal speed - precisely nullifying gravity - allows a quadrocopter to stay in the air without moving. The following actions can be taken to maneuver the quadrocopter:

- vertical acceleration is achieved by increasing or decreasing the speed of all four rotors equally,
- yaw rotation can be achieved by increasing the speed of engines 1 and 4, while decreasing the speed of engines 2 and 3 (or vice-versa) - resulting in an overall clockwise (or counter-clockwise) torque, without changing overall upwards thrust or balance,
- horizontal movement can be achieved by increasing the speed of one engine, while decreasing the speed of the opposing one, resulting in a change of the roll or pitch angle, and thereby inducing horizontal acceleration.

The fine tuning of the relative engine speeds is very sensible to small changes, making it difficult to control a quadrocopter without advanced controlling routines and accurate sensors.

2.1.2. The Parrot AR.Drone

The Parrot AR.Drone has dimensions of 52.5 cm $\times$ 51.5 cm with, and 45 cm $\times$ 29 cm without hull. It has four rotors with a 20 cm diameter, fastened to a robust carbon-fiber skeleton cross providing stability. A removable styrofoam hull protects the drone and particularly the rotors during indoor-flights, allowing the drone to survive minor and not-so-minor crashes such as flying into various types of room furniture, doors and walls - making it well suited for experimental flying and development. An alternative outdoor-hull - missing the rotor-protection and hence offering less protection against collisions - is also provided and allows for better maneuverability and higher speeds.

The drone weights 380 g with the outdoor-hull, and 420 g with the indoor-hull. Although not officially supported, in our tests the drone was able to fly with an additional payload of up to 120 g using the indoor hull - stability, maneuverability and battery life however suffered significantly, making the drone hardly controllable with that kind of additional weight.

The drone is equipped with two cameras (one directed forward and one directed downward), an ultrasound altimeter, a 3-axis accelerometer (measuring acceleration), a 2-axis gyroscope (measuring pitch and roll angle) and a one-axis yaw precision gyroscope. The

Figure 2.2. The Parrot AR.Drone with indoor-hull (left) and outdoor-hull (right).

onboard controller is composed of an ARM9 468 MHz processor with 128 Mb DDR Ram, on which a BusyBox based GNU/Linux distribution is running. It has an USB service port and is controlled via wireless LAN.

Cameras

The AR.Drone has two on-board cameras, one pointing forward and one pointing downward. The camera pointing forward runs at 18 fps with a resolution of 640×480 pixels, covering a field of view of $73.5° \times 58.5°$. Due to the used fish eye lens, the image is subject to significant radial distortion. Furthermore rapid drone movements produce strong motion blur, as well as linear distortion due to the camera's rolling shutter (the time between capturing the first and the last line is approximately 40 ms).

The camera pointing downwards runs at 60 fps with a resolution of 176×144 pixels, covering a field of view of only $47.5° \times 36.5°$, but is afflicted only by negligible radial distortion, motion blur or rolling shutter effects. Both cameras are subject to an automatic brightness and contrast adjustment.

Gyroscopes and Altimeter

The measured roll and pitch angles are, with a deviation of only up to 0.5°, surprisingly accurate and not subject to drift over time. The yaw measurements however drift significantly over time (with up to 60° per minute, differing from drone to drone - much lower values have also been reported [17]). Furthermore an ultrasound based altimeter with a maximal range of 6 m is installed on the drone.

2.2. Software

The Parrot AR.Drone comes with all software required to fly the quadcopter. Due to the drone being a commercial product which is primarily sold as high-tech toy and not as a

tool for research, accessing more than this basic functionality however turns out not to be so easy.

The first and most important drawback is, that the software running onboard is not accessible: while some basic communication via a telnet shell is possible, the control software is neither open-source nor documented in any way - while custom changes including starting additional processes are possible, this would require massive trial and error and is connected with a risk of permanently damaging the drone [2]. For this thesis we therefore treat the drone as a black box, using only the available communication channels and interfaces to access its functionalities.

This section is dedicated to describing these communication channels and the provided functionalities, as well as introducing the official SDK for controlling the drone from any operation system.

2.2.1. Communication Channels

As soon as the battery is connected, the drone sets up an ad-hoc wireless LAN network to which any device may connect. Upon connect, the drone immediately starts to communicate (sending data and receiving navigational commands) on four separate channels:

- navigation channel (UDP port 5554),
- video channel (UDP port 5555),
- command channel (UDP port 5556),
- control port (TCP port 5559, optional).

Note that the three major communication channels are UDP channels, hence packages may get lost or be received in the wrong order. Also note the complete lack of any security measures - anybody may connect to and control a running drone at any time, no password-protection or encryption is possible.

Navigation Channel

While in normal mode the drone only broadcasts basic navigational data every 30 ms, after switching to debug mode it starts sending large amounts of sensor measurements every 5 ms. The exact encoding of the sent values will not be discussed here, it is partially documented in [32]. The most important parameters and sensor values - and the ones used in our approach - are the following:

- drone orientation as roll, pitch and yaw angles: as mentioned in the previous section, roll and pitch values are drift-free and very accurate, while the measured yaw-angle is subject to significant drift over time,
- horizontal velocity: in order to enable the drone to keep its position in spite of wind, an optical-flow based motion estimation algorithm utilizing the full 60 fps from the floor camera is performed onboard, estimating the drone's horizontal speed. The exact way these values are determined however is not documented.

Experiments have shown that the accuracy of these values strongly depends on whether the ground below the drone is textured or not: when flying above a textured surface (or, for example, a cluttered desk) these values are extremely accurate - when flying above a poorly textured surface however, the quality of these speed estimates is very poor, deviating from the true value by up to $1\,\mathrm{m/s}$ above completely untextured surfaces.

- drone height in millimeter: this value is based solely on the ultrasound altimeter measurements. As long as the drone is flying over a flat, reflecting surface, this value is quite accurate, with (at a height of 1 m) a mean error of only $8,5\,\mathrm{cm}$. As this sensor measures relative height, when flying over uneven surfaces or close to walls strong fluctuations will occur. This often induces sudden and undesired vertical acceleration, as the drone tries to keep its relative height as supposed to its absolute height. This value is measured only every 40 ms.
- battery state as an integer between 100 and 0,
- the control state as a 32-bit bit field, indicating the drone's current internal status. This might for example be "LANDED", "HOVERING", "ERROR", "TAKEOF" etc.,
- the drone's internal timestamp, at which the respective data was sent, in microseconds. This is not necessarily the time at which the sensor values were taken, experiments have shown that within the same package, some parameters are up to 60 ms older than others.

Video Channel

The drone continuously transmits one video stream, which can be one of four different channels - switching between channels can be accomplished by sending a control command to the drone. The four available channels are depicted in Figure 2.3. As can be seen, neither of the available cameras can be accessed fully: for the downwards facing camera the available frame rate is - with only 18 fps - significantly lower than the original 60 fps. Furthermore the maximal supported resolution is 320×240, halving the forward camera's original resolution[1].

The video stream is encoded using a proprietary format, based on a simplified version of the H.263 UVLC codec [41]. Images are encoded in YCBCR color space, 4:2:0 type[2], using 8 bit values. More details can be found in [32]. While the achieved compression is fairly good (in practice around 10 kB per frame, resulting in a bandwidth required of only 180 kBps), this encoding produces significant artifacts in the decoded picture.

Command Channel

The Drone is navigated by broadcasting a stream of command packages, each defining the following parameters:

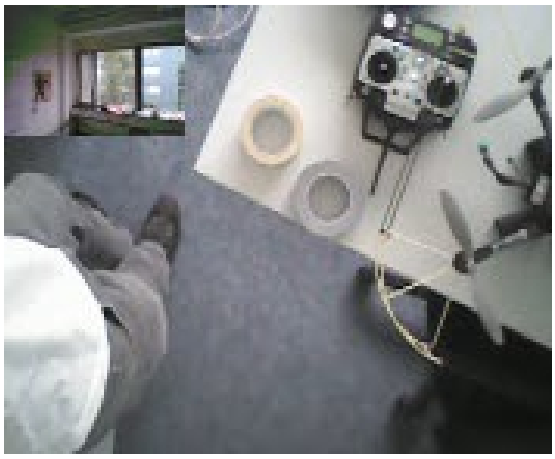
[1] In earlier drone firmware versions, the full 480×640 resolution was available for channels 1 and 2. This however has been changed, because encoding a 480×640 video stream puts too much computational load on the on-board processor, occasionally affecting other vital functionalities such as flight stabilization.

[2] the cb and the cr channel are only sampled at half the original resolution

(a): channel 1 (320 × 240 @ 18 fps)

(b): channel 2 (320 × 240 @ 18 fps)

(c): channel 3 (176 × 144 @ 18 fps)



(d): channel 4 (176 × 144 @ 18 fps)

Figure 2.3. The four different video channels available: (a) frontal camera only, (b) frontal camera with embedded floor camera (87 × 72), (c) bottom camera only, (d) bottom camera with embedded frontal camera (58 × 42).

Figure 2.4. The AR.FreeFlight App running on an iPhone. While for horizontal movement the device is simply tilted in the respective direction, altitude and yaw rotation are controlled using the button on the right side.

1. desired roll and pitch angle, yaw rotational speed as well as vertical speed, each as fraction of the allowed maximum, i.e. as value between -1 and 1,

2. one bit switching between hover-mode (the drone tries to keep its position, ignoring any other control commands) and manual control mode,

3. one bit indicating whether the drone is supposed to enter or exit an error-state, immediately switching off all engines,

4. one bit indicating whether the drone is supposed to take off or land.

Being sent over an UDP channel, reception of any one command package cannot be guaranteed. In our implementation the command is therefore re-sent approximately every 10 ms, allowing for smoothly controlling the drone.

Control Port

Control commands can be used to change internal settings of the drone, for example for switching between the four available video channels. In general a control command is transmitted as a string of the format "[attribute]=[value]", for a list of the available commands we refer to [32].

2.2.2. Controlling the AR.Drone via iPone

Controlling the AR.Drone using a smart phone or tablet PC is straight-forward using the freely available, open source AR.FreeFlight App provided by Parrot. While sufficient for merely flying the drone, this App has some shortcomings - for example it is not possible to record a video of the drone's camera feed or to change internal settings of the drone.

2.2.3. Available Software

Parrot provides a software development kit (SDK) to facilitate connecting to the drone from any operating system. The SDK consists of a multi platform core written in plain ANSI C code, as well as basic example applications for various platforms including Windows, Linux, iPhone and Android. The SDK core takes care of the following tasks:

1. setting up the drone's communication channels,
2. receiving and decoding video frames, calling a specified callback function every time a new frame is available,
3. receiving and decoding navigational data, calling a specified callback function every time new sensor data is available,
4. encoding and sending control commands.

The SDK is partially documented in [32], however a large part is neither documented nor commented, making it difficult to use or modify. Furthermore the Windows and Linux examples contain a number of bugs, the Windows example even raises a number of compilation errors and does not reset properly after a connection loss.

3. Monocular SLAM

Simultaneous localization and mapping (SLAM) is the process of continuously estimating the position and orientation of a robot in a previously unknown environment. This is achieved by incrementally building a map of the environment from the available sensor data, which, at the same time, is used to re-estimate the position of the robot in regular intervals. SLAM systems are a key component of autonomously acting robots, and are a requirement for navigation in a previously unknown environment. Furthermore, such methods are the basis for many augmented reality (AR) applications, allowing to project additional, virtual components into a video as if they were part of the real scene.

SLAM systems can employ different kinds of sensors: non-visual ones such as ultrasound or a laser-range scanner, or visual ones such as for example a monocular camera, a stereo-camera or a red-green-blue-depth (RGBD) camera, providing not only the color but also the depth of every pixel such as Microsoft's Kinect or time-of-flight cameras. While the mathematical problem behind SLAM is similar for different types of sensors, sensors that provide more information can greatly simplify the process and reduce the computational cost. Particularly the possibility of measuring depth (using stereo or RGBD-cameras) eliminates several challenges encountered when relying only on a monocular camera - however often only a monocular camera is available. One particular drawback of all depth-measuring devices is the very limited range at which they can operate accurately - in order to navigate in large, open spaces (e.g. a factory building, or outside in GPS-denied environments such as narrow streets), monocular SLAM systems are essential. In this thesis we therefore focus on monocular SLAM, that is SLAM based on a single camera moving through three-dimensional space.

In visual SLAM systems, the map of the environment is typically represented by a number of landmarks, i.e. points in three-dimensional space that can be recognized and localized in the camera image, typically appearing as small, distinctive regions or patches (keypoints). Based on the locations of these keypoints in the image, the position of the camera can be estimated. As new parts of the environment become visible, additional landmarks are identified, added to the map and can then be integrated into the pose-estimation process.

Although the SLAM problem has received significant attention from researchers in the past decades, several open challenges remain. Particularly dealing with large or dynamic environments, keeping computational complexity feasible while the map is growing, minimizing global drift and efficient detection of loop-closures are subject to current research. One particular problem of monocular SLAM is the inherent scale ambiguity: due to the projective nature of the sensor, map and movement of the camera can only be determined up to scale - without a calibration object of known dimensions or additional sensor information, one degree of freedom remains undetermined.

In this chapter, we first give a short historical introduction to the SLAM problem and summarize the current state of the art in Section 3.1, in particular we outline differences and similarities between two fundamentally different approaches to the problem: the filtering-based and the keyframe-based approach. We then give a detailed description of a keyframe-based monocular SLAM system. First, in Section 3.2, we introduce the notation and conventions used in this chapter. A brief outline of the complete system is given in Section 3.3, in particular we identify the three main components initialization, tracking and mapping. We then proceed to presenting the concept of keypoints and landmarks in Section 3.4, and how they can be detected and tracked in an image. In the subsequent three sections, we give a detailed mathematical description of the three identified components: initialization in Section 3.5, mapping in Section 3.6 and tracking in Section 3.7. In Section 3.8, we give a short summery of the contents of this chapter.

3.1. State of the Art

Historically, the SLAM problem has been formulated first in 1986 [7], and is widely considered to be one of the key prerequisites of truly autonomous robots [28]. First proposed solutions were based on the extended Kalman filter (EKF), these methods are also known as **EKF-SLAM**: Positions of landmarks as well as the current position of the robot are jointly represented as current state vector $\mathbf{x}$. The fundamental problem of this approach is the fact that the computational complexity of incorporating an observation, which is required for each new pose-estimate, scales quadratically in the number of landmarks (as a full update of the maintained covariance matrix of $\mathbf{x}$ is required for each step). This limits the number of landmarks to a few hundred in practice, making it impossible to navigate in larger environments.

This limitation was addressed by **FastSLAM** by Montemerlo et al. [28]: The observation that conditioned on the robot's path, the positions of the individual landmarks become independent allows for each landmark's position to be estimated independently. Using a particle filter instead of a Kalman filter (each particle representing one possible path taken and maintaining its own estimate of all landmark positions), leads to a naïve complexity for each update of $O(kn)$, k being the number of particles and n the number of landmarks. Using a tree-based data structure, this can be reduced to $O(k \log n)$, hence logarithmic instead of quadratic complexity in the number of landmarks - rendering maps containing tens of thousands of landmarks computationally feasible.

The emergence of keyframe-based methods such as **parallel tracking and mapping** (PTAM) by G. Klein and D. Murray in 2007 [14] can be seen as a major milestone in the development of monocular SLAM methods. Keyframe-based approaches differ in many ways from previously dominant filtering-based approaches such as EKF-SLAM or FastSLAM: Instead of marginalizing out previous poses and summarizing all information within a probability distribution (filtering), keyframe-based approaches retain a selected subset of previous observations - called keyframes - explicitly representing past knowledge gained. This difference becomes particularly evident when looking at how the map is represented (see also Figure 3.1):

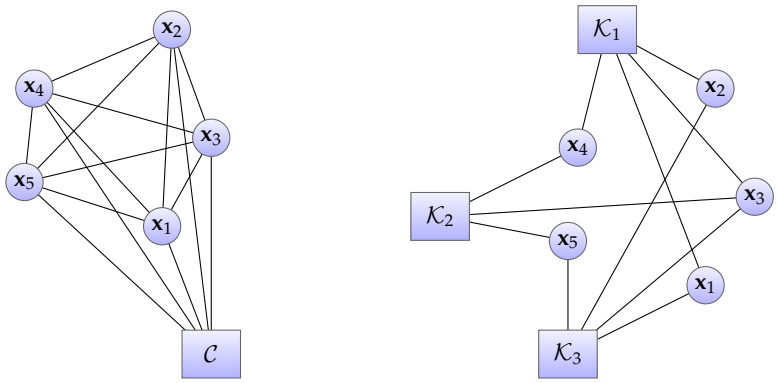


Figure 3.1. (a) Visual representation of the map for filtering-based approaches: correlations between landmark positions are explicitly represented as covariance, and the current camera position is an integral part of the map. The map can be interpreted as a fully connected graph with $\{\mathbf{x}_i\} \cup \{\mathcal{C}\}$ as nodes. (b) Visual representation of the map for keyframe-based approaches: observations serve as link between landmarks and keyframes, correlations between landmarks are not explicitly represented. The map can be interpreted as a bipartite graph, with $\{\mathbf{x}_i\} \cup \{\mathcal{K}_j\}$ as nodes and all observations as edges.

Map Representation for Filtering-Based Approaches

- joint probability distribution over landmark-positions $\mathbf{x}_1 \dots \mathbf{x}_n \in \mathbb{R}^3$ and current camera position $\mathcal{C} \in \mathrm{SE}(3)$, e.g. as multivariate Gaussian distribution with mean $\mathbf{x} := (\mathbf{x}_1, \dots, \mathbf{x}_n, \mathcal{C}) \in \mathbb{R}^{3n+6}$ and covariance $\mathbf{\Sigma} \in \mathbb{R}^{(3n+6)\times(3n+6)}$
- appearance of landmarks as image-patches

Map Representation for Keyframe-Based Approaches

- positions of landmarks $\mathbf{x}_1 \dots \mathbf{x}_n \in \mathbb{R}^3$
- positions of keyframes $\mathcal{K}_1 \dots \mathcal{K}_m \in \mathrm{SE}(3)$ and respective camera images $I_1 \dots I_m$
- all landmark observations in keyframes $\bar{\mathbf{p}}_{ij} \in \mathbb{R}^2$ for $i = 1 \dots n$ and $j = 1 \dots m$

Based on this keyframe-based map representation, the process of simultaneous localization and mapping can be split into two distinct tasks:

1. **Tracking**: estimating the position of the camera $\mathcal{C}$ based on a fixed map, using only the landmark positions $\mathbf{x}_i$.
2. **Mapping**: optimizing keyframe and landmark positions $\mathbf{x}_i$ and $\mathcal{K}_j$ such that they coincide with the observations $\bar{\mathbf{p}}_{ij}$. This process is called bundle adjustment (BA), and is detailed further in Section 3.6. Furthermore, new keyframes and landmarks have to be integrated into the map, and other modifications such as removing invalid landmarks, observations or even keyframes are applied.

In the last years, these two approaches have become more and more interconnected, with recent publications such as [13, 16, 24] adopting the advantages of both approaches. Strasdat et al. [39] have shown both by theoretical considerations as well as practical comparisons, that in general keyframe-based approaches are more promising than filtering. More precisely, they argue that increasing the number of landmarks is more beneficial than incorporating more observations in terms of accuracy gained per unit of computation.

Particularly the second task - optimizing a map consisting of keyframes, landmarks and observations - has received significant attention: Strasdat et al. have explored different bundle adjustment strategies [37] and proposed large scale and loop closure methods [38]. Kümmerle et al. developed a general framework for graph optimization called g^2o [18], generalizing the idea of bundle adjustments: In particular, this framework allows to include explicit constraints on the relationship between two keyframes (relative rotation and translation) into the bundle adjustment process, allowing for truly constant-time monocular SLAM. Using this approach, the size of the mapped area is only constrained by the memory required to store the map, and not by computational power available. The key idea behind this is to enforce the map to be euclidean only locally, while globally it is treated as a topological map [37].

Dense SLAM

In the last years, novel dense SLAM methods - both, based on monocular cameras as well as based on RGBD-cameras - have been presented [29, 11]. These methods not only build a dense map of the environment in form of a closed surface, but also use the whole image for tracking - to some extend eliminating the need for keypoints and well-textured objects. All dense methods however heavily rely on massive parallelization utilizing modern GPU hardware, restricting their use for mobile robots. Furthermore, the map is stored in a three-dimensional voxel cube, limiting the maximal map size to a small area[1]. This problematic might be addressed in the near future by the use of octrees, as only a small portion of the voxels (close to the surface boundaries, approx. 15%) contain relevant information.

3.2. Notation

We adopt the following conventions:

- matrices are denoted by upper-case, bold letters and (column-)vectors by lower-case, bold letters,
- an image or a camera frame is interpreted as a continuous function $I(x, y) : \Omega \to \mathbb{R}$ where $\Omega \subset \mathbb{R}^2$ is the image space, mapping pixel coordinates to brightness values,
- the homogeneous representation of a point $\mathbf{x} \in \mathbb{R}^d$ is denoted by $\tilde{\mathbf{x}} := (\mathbf{x}^T, 1)^T \in \mathbb{R}^{d+1}$,

[1] In the original version of DTAM as described in [29], the map consisted of a collection of 2.5 dimensional keyframes instead of a global voxel cube - it did however not contain a dense equivalent to bundle adjustment. A newer version, presented as demo at the ICCV 2011 however stores all map information in a global voxel cube, similar to KinectFusion [11].

- dehomogenization and perspective projection is denoted by the function $\text{proj}\colon \mathbb{R}^{d+1} \to \mathbb{R}^d$ defined by $\text{proj}(x_1 \dots x_{d+1}) := x_{d+1}^{-1}(x_1 \dots x_d)^T$,
- coordinate systems are denoted by calligraphic, upper-case letters. In particular, $\mathcal{W}$ denotes the world-coordinate system, $\mathcal{C}$ the camera-coordinate system and $\mathcal{K}_j$ the camera-coordinate system at keyframe j. If required, the coordinate system a point is expressed in is indicated by adding it as subscript.
- the matrix transforming points from a coordinate system $\mathcal{K}_1$ to a different coordinate system $\mathcal{K}_2$ is denoted by $\mathbf{E}_{\mathcal{K}_2\mathcal{K}_1} \in \text{SE}(3) \subset \mathbb{R}^{4\times4}$ ($\mathbf{x}_{\mathcal{K}_2} = \text{proj}(\mathbf{E}_{\mathcal{K}_2\mathcal{K}_1}\tilde{\mathbf{x}}_{\mathcal{K}_1})$). For transformation matrices from the world coordinate system, we abbreviate $\mathbf{E}_{\mathcal{K}_1} := \mathbf{E}_{\mathcal{K}_1\mathcal{W}}$.
- the camera projection matrix is denoted by $\mathbf{K}_{\text{cam}} \in \mathbb{R}^{3\times3}$,
- landmark positions expressed in the world-coordinate system are denoted by $\mathbf{x}_i \in \mathbb{R}^3, i = 1 \dots n$ with n being the number of landmarks,
- an observation of landmark i in keyframe j is denoted by $\mathbf{p}_{ij} \in \mathbb{R}^2$, in normalized image coordinates. The respective pixel-coordinates are denoted by $\tilde{\mathbf{p}}_{ij} := \text{proj}(\mathbf{K}_{\text{cam}}\tilde{\mathbf{p}}_{ij})$.

When optimizing over a rotation matrix $\mathbf{R} \in \text{SO}(3) \subset \mathbb{R}^{3\times3}$, it is assumed to be parametrized as a rotation vector $\mathbf{r} \in \mathfrak{so}(3) \subset \mathbb{R}^3$, using Rodrigues' formula. When optimizing over a transformation matrix between coordinate systems $\mathbf{E} \in \text{SE}(3) \subset \mathbb{R}^{4\times4}$, it is assumed to be parametrized as translation vector plus rotation vector, i.e. $\begin{pmatrix}\mathbf{t}\\ \mathbf{r}\end{pmatrix} \in \mathbb{R}^6$.

3.3. Monocular, Keyframe-Based SLAM: Algorithm Outline

As mentioned in the introduction, the SLAM process can be split into two distinct parts, running independently: **tracking** and **mapping**. As both parts require an existing map to operate on, a separate **initialization procedure** generating an initial map is required. The interaction between these three components is visualized in Figure 3.2.

- **Initialization** is performed once after the algorithm is started and requires a certain type of camera-motion, e.g. translation parallel to the image plane.
  input: initial video frames
  output: initial map
- **Mapping** runs in a continuous loop, constantly optimizing the map and integrating new keyframes when instructed to by the tracking component.
  input: the whole map, new keyframes to integrate
  output: updated map
- **Tracking** runs in a continuous loop which is evaluated once for each new video frame.
  input: new video frame I, landmark positions $\mathbf{x}_1 \dots \mathbf{x}_n$ within the map
  output: camera pose $\mathcal{C}$ at this frame

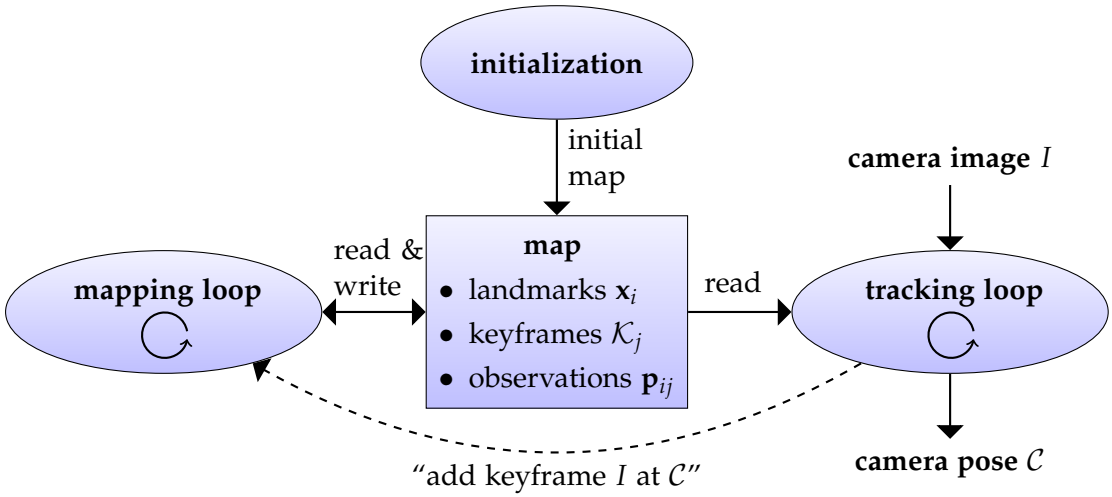


Figure 3.2. Schematic outline of a keyframe-based SLAM algorithm. Three main parts can be identified: initialization, tracking and mapping. While the initialization stage only generates the initial map, tracking and mapping are performed simultaneously in two continuous loops, both operating on the same map.

In keyframe-based, monocular SLAM algorithms (here, as an example, from PTAM), the map consists of:

- for each keyframe $\mathcal{K}_j$ the transformation from the global coordinate system $\mathbf{E}_{\mathcal{K}_j}$ and the image I_j itself,
- for each landmark its world-coordinates $\mathbf{x}_i \in \mathbb{R}^3$, the estimated normal of the patch $\mathbf{n}_i \in \mathbb{R}^3$ and a reference to its source pixels,
- the locations of all landmark-observations in keyframes $\bar{\mathbf{p}}_{ij} \in \mathbb{R}^2$, and the respective observation uncertainties $\sigma_{ij}^2 = 2^l$, where l is the scale at which the landmark was observed (the coarser the keypoint i.e. the bigger the image patch, the larger the uncertainty).

3.4. Keypoints

Keypoints - also called local feature points or points of interest - are the basic building block of a vast amount of methods in the field of computer vision. They are motivated by the fact that processing the image as a whole is computationally unfeasible - instead a small set of particularly "interesting" and distinguishable image segments is used for tasks such as object recognition, detection and tracking, pose estimation, image registration and many more. Keypoints are small, distinguishable areas (windows) in the image, that are frequently assumed to be locally planar segments of objects in the scene, and therefore mainly occur on strongly textured objects. Note that this is only one of many possible approaches: particularly edge-based (Canny edge detector [5], edgelets [8]) or region-based (MSER [23], PCBR [6]) features have also been explored and used in SLAM systems [8].

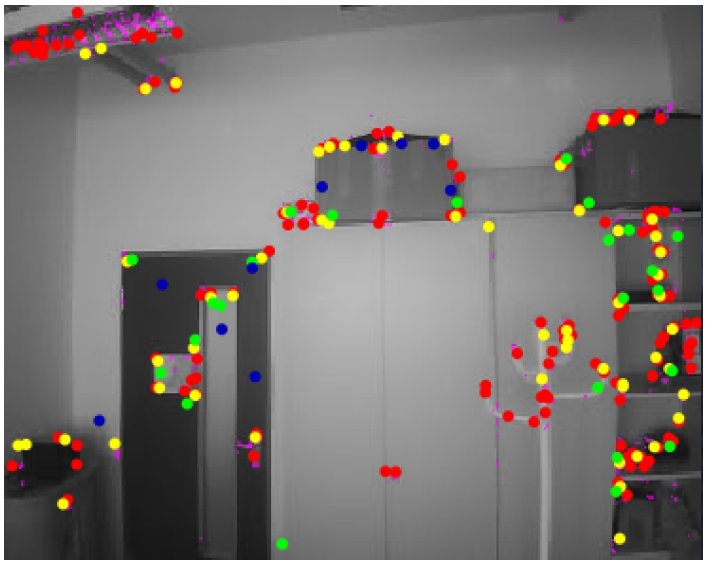


Figure 3.3. Locations of keypoints used by PTAM for tracking. The color represents the pyramid-level on which the keypoints was found: blue dots correspond to coarse and large patches, while red dots correspond to very fine and small keypoints.

In this thesis, a keypoint is a two-dimensional location in the camera image, while the corresponding three-dimensional position will be referred to as a landmark. Keypoints are hence the two-dimensional projections of landmarks onto the image plane. In the remainder of this section, we first explain the characteristics of good keypoints and how they can be formulated mathematically, which leads to the well-known Harris corner-detector in Section 3.4.1. Furthermore we present the FAST corner detector and the Laplacian of a Gaussian blob detector. In Section 3.4.2, we briefly introduce the concept of scale of a keypoint. We then present a simple method to track keypoints in a video in Section 3.4.3.

3.4.1. Identifying Good Keypoints

One of the earliest and most widely used keypoint detectors was presented by Harris and Stephens in 1988 [9], and is called the Harris corner detector. The basic idea is to define an image-region (patch) as good for tracking, if shifts in *all* directions can be detected easily. Mathematically this can be formulated by demanding that the change in appearance when shifting the window by (x, y) be large for every (x, y) on the unit circle. This change in appearance can be measured by a weighted sum of squared differences (SSD), and is denoted by $S(x, y)$. As weighting-function $w(u, v)$, usually a Gaussian kernel with fixed variance σ^2 is used:

$$S(x,y) := \sum_{u,v} w(u,v)(I(u+x, v+y) - I(u,v))^2 \tag{3.1}$$

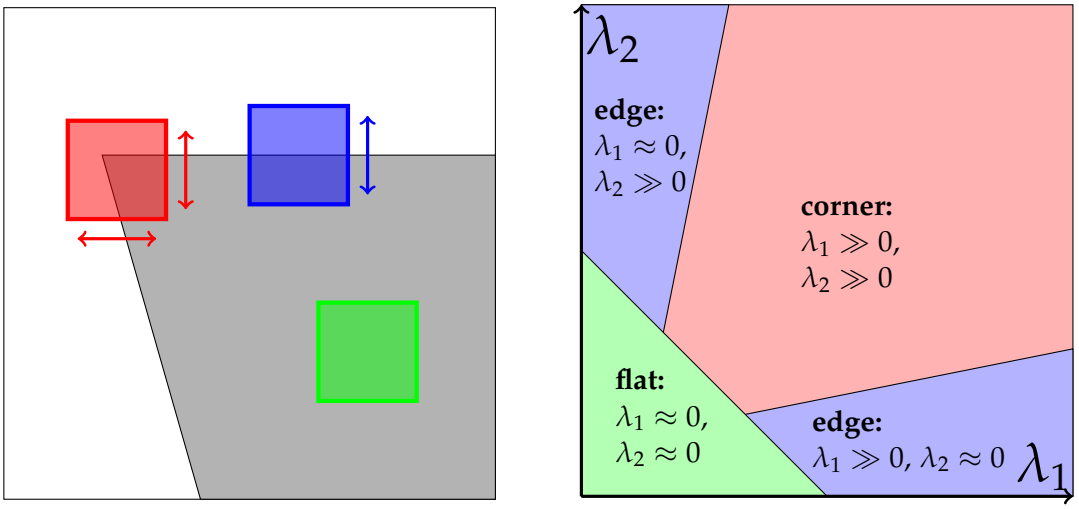


Figure 3.4. The three different patch types that can be distinguished: for a corner (red), bot eigenvalues are large and shifts in any direction can be detected. For an edge (blue), one eigenvalue is large while the other is small, only shifts perpendicular to the edge can be detected. For a flat region (green), both eigenvalues are small and no shifts can be detected.

Approximating $I(u+x, v+y)$ by a first-order Taylor expansion, this can be expressed in terms of the spacial derivatives of the image, I_x and I_y:

$$S(x,y) \approx \sum_{u,v} w(u,v)(I_x(u,v)\cdot x + I_y(u,v)\cdot y)^2 \tag{3.2}$$

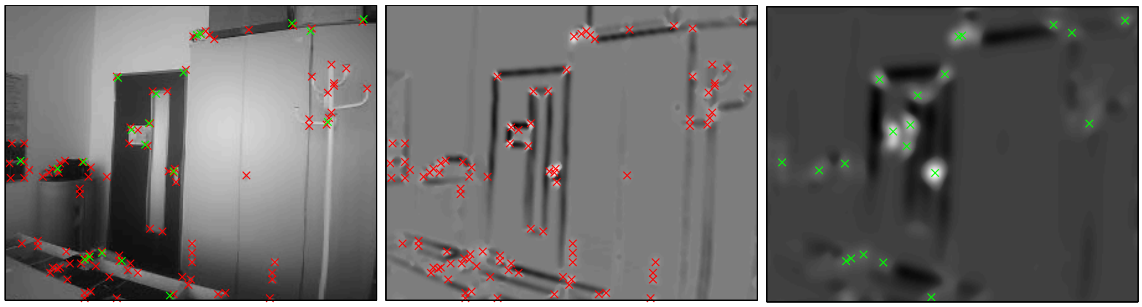
Which can be simplified to:

$$\begin{aligned} S(x,y) &\approx (x,y)\mathbf{A}\begin{pmatrix}x\\y\end{pmatrix} \\ \text{with}\quad \mathbf{A} &:= \sum_{u,v} w(u,v)\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \end{aligned} \tag{3.3}$$

Note that in this derivation, location and dimensions of the patch examined are implicitly encoded in the weighting function, i.e. as mean μ and variance σ^2 in case of a Gaussian kernel. Due to $\mathbf{A}$ (also called the structural tensor of a patch) being symmetric and positive semi-definite, the two extrema $\max_{x,y} S(x,y)$ and $\min_{x,y} S(x,y)$ turn out to be the two eigenvalues of $\mathbf{A}$, denoted by λ_1 and λ_2. In fact, it can be observed that the nature of a patch can be characterized by looking at the eigenvalues of $\mathbf{A}$ alone, yielding an efficient algorithm to find good keypoints in an image (see Figure 3.4).

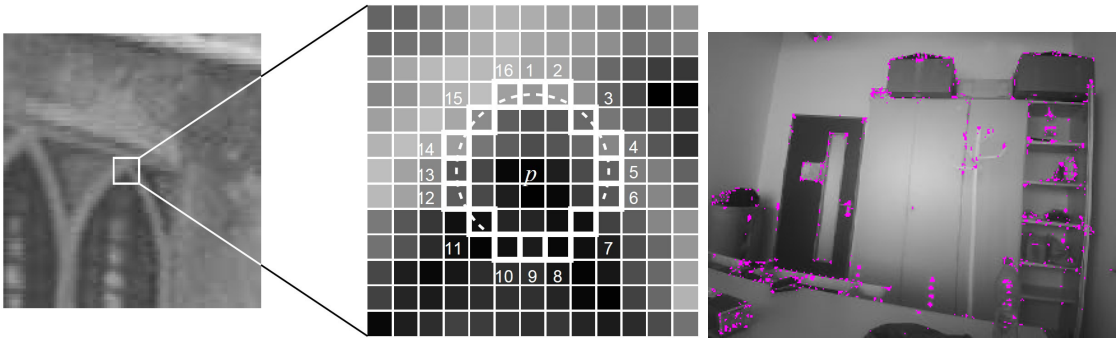
In their original paper, Harris and Stephens considered the exact computation of λ_1 and λ_2 to be computationally to expensive and instead suggested the following response function:

$$\begin{aligned} M_c &:= \lambda_1\lambda_2 - \kappa\,(\lambda_1+\lambda_2)^2 \\ &= \det(\mathbf{A}) - \kappa\ \mathrm{trace}^2(\mathbf{A}) \end{aligned} \tag{3.4}$$



(a) (b) (c)

Figure 3.5. The Harris response function M_c: (a) original image, (b) visualization of M_c for $\sigma^2 = 2.25$, (c) visualization of M_c for $\sigma^2 = 25$.



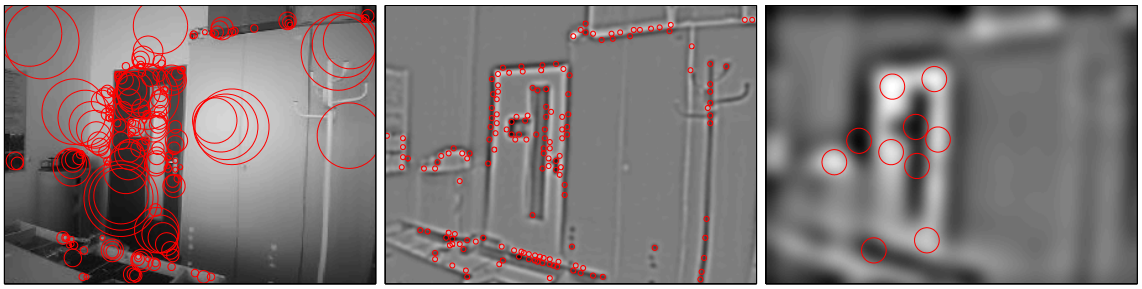
(a) (b)

Figure 3.6. (a) The FAST-16 corner detector inspects a circle of 16 pixels. As 12 continuous pixels brighter than the center are found, the patch is classified as a corner. (b) FAST corners without nonmaximum-suppression. Processing an image of 320×240 pixel takes less than 1 ms on modern hardware.

with κ being a positive constant. Shi and Tomasi later showed that using $\min(\lambda_1, \lambda_2)$ as response function is more stable, the resulting detector is also referred to as Kanade-Tomasi corner detector [34]. The full detector now operates by calculating M_c at each pixel and choosing keypoints at local maxima which exceed a certain threshold.

The FAST Corner Detector

As the acronym already suggests, the FAST corner detector, presented by Rosten and Drummond in 2006 [33], has the advantage of being significantly faster than other methods, achieving a speedup of factor 20 to 30 compared to Harris corners. The key idea is to explicitly search for corners, deciding whether a pixel is the center of a corner by looking at pixels aligned in a circle around it: if a long enough sequence of continuously brighter or continuously darker pixels is found, it is classified as a corner. Rosten and Drummond proposed to learn a decision tree to decide for each pixel in an image whether it is a corner or not, using as few comparisons as possible. The learned decision trees are available as C code, consisting of one gigantic if-then-else construct spanning over thousands of lines. Although significantly faster, in several ways FAST performs as good as or even better



(a) (b) (c)

Figure 3.7. Result of the Laplace response function M_L: (a) original image and blobs found on different scales, (b) $M_L(x, y; 2)$ and respective blobs, (c) $M_L(x, y; 10)$ and respective blobs.

than other corner detectors, and often is the method of choice for time-critical applications such as real-time SLAM systems.

The Laplacian of a Gaussian Blob Detector

The Laplacian of Gaussian (LoG) blob detector [27, 21] finds blobs in the image instead of corners, these are approximately circular regions in the image that are darker or brighter than the surrounding. Such blobs are found by identifying local extrema of the LoG response function $M_L(x, y; t)$, which is the Laplacian of the smoothed image:

$$\begin{aligned} M_L(x, y; t) &:= t \cdot \Delta L(x, y; t) \\ \text{with } L(x, y; t) &:= g(x, y; t) * I(x, y) \end{aligned} \tag{3.5}$$

$g(x, y; t)$ denotes a Gaussian kernel with variance t. Due to the associativity of convolution, this can efficiently be calculated by convolving the image with two one-dimensional filters, namely the Laplacian of a one-dimensional Gaussian in both dimensions. $M_L(x, y; t)$ produces strong local extrema for blobs with radius $\sqrt{t}$ - in order to find blobs of different sizes, it has to be calculated for different values of t. Analytically it can be shown that $L(x, y; t)$, the so-called scale-space representation of an image satisfies the diffusion equation:

$$\partial_t L = \frac{1}{2} \Delta L \tag{3.6}$$

allowing $M_L(x, y; t)$ to be approximated by a difference of Gaussians (DoG):

$$M_L(x, y; t) \approx \frac{t}{\Delta t} \left(L(x, y; t + \Delta t) - L(x, y; t - \Delta t) \right) \tag{3.7}$$

As for many algorithms $L(x, y; t)$ for different values of t needs to be computed anyway, M_L can be computed by simply subtracting two images, making it computationally inexpensive.

3.4.2. Multiscale Keypoint Detection

When applying a corner detector such as FAST or Harris, only corners of a certain size are detected. In order to identify keypoints at different scales, these techniques are combined with a multiscale approach: the image is downscaled or blurred generating an image pyramid, while the relative patch size is increased, allowing for "larger" corners to be detected.

A popular approach, used for example in the well-known Lucas-Kanade tracker (LKT), is to detect Harris corners at each pyramid-level, and remove points that do not correspond to a local maximum in scale-dimension with respect to their DoG-response. Practical results showed that with respect to the LKT, Harris corners perform best in spacial dimensions, while the DoG response function provides a more stable maximum in scale dimension [27, 21].

3.4.3. Tracking a Keypoint

Tracking a keypoint is the task of finding the exact position (and possibly other parameters such as scale and orientation) of a keypoint in an image, *under the assumption that a good initialization is available*. When processing for example a video, the position of the patch in the previous frame can be used as initialization, assuming that the displacement between two consecutive frames is small. Tracking is not to be confused with methods for matching keypoints such as SIFT [22] and SURF [3] (which build an invariant representation of a patch) or Ferns [31] (training a classifier to recognize a patch from any viewpoint), although these methods can also be used for tracking, referred to as tracking by detection. A general formulation of tracking is to find parameters $\mathbf{p}$ of a warp function $f(x, y; \mathbf{p})\colon \mathbb{R}^2 \times \mathbb{R}^d \to \mathbb{R}^2$, such that the difference between the original patch $T(x, y)$ and the transformed image $I(f(x, y; \mathbf{p}))$ becomes minimal, that is minimizing the sum of squared differences (SSD):

$$\begin{aligned} \mathbf{p}^* &= \arg\min E_{\text{SSD}}(\mathbf{p}) \\ \text{with } E_{\text{SSD}}(\mathbf{p}) &:= \sum_{x,y} (I(f(x, y; \mathbf{p})) - T(x, y))^2 \end{aligned} \tag{3.8}$$

In order to achieve invariance to affine lightning changes, a normalized cross-correlation error function can be used instead. Note that this error function is highly non-convex and hence only a local minimum can be found, underlining the need for a good initialization. The warp function $f(x, y; \mathbf{p})$ can take different forms, for tracking a two-dimensional image patch two important warp functions are:

Pure Translation: It is often sufficient to consider only translation for frame-to-frame tracking. The resulting transformation has two degrees of freedom, the displacement in two dimensions:

$$f(x, y; \delta x, \delta y) = \begin{pmatrix} x + \delta x \\ y + \delta y \end{pmatrix} \tag{3.9}$$

Affine Transformation: An affine transformation allows for displacement, non-uniform

scaling and rotation, leading to 6 degrees of freedom:

$$f(x, y; \mathbf{p}) = \begin{pmatrix} p_1 & p_2 \\ p_3 & p_4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} p_5 \\ p_6 \end{pmatrix} \tag{3.10}$$

For practical applications, choosing a good warp function and parametrization for the task at hand is essential.

3.4.4. Summary (Keypoints)

As keypoints are a fundamental component of many algorithms in the field of computer vision, a wide variety of methods is available, of which only a small portion has been discussed in this section. It can be summarized that the Harris corner detector, combined with a LoG-based scale-selection performs best when computationally feasible, allowing for a higher scale-resolution and yielding accurate and robust results. In time-critical applications - as for example a SLAM system - the FAST corner detector has the advantage of being significantly faster, however due to its discrete nature only a small number of scales can be used (halving the image resolution for each scale).

For frame-to-frame tracking, a purely translational warp function is sufficient, as changes in scale and perspective are small. When using a fixed template however - for example from a reference image of an object of interest - an affine, or even a projective warp function is required.

The SLAM system used in this thesis (PTAM) uses FAST keypoints on four different scales. Keypoints are tracked with respect to translation only, using a keyframe the keypoint was observed in as template. In order to compensate for the potentially large change in viewpoint, the template is warped according to the expected change in viewpoint prior to the tracking process - making use of the fact that the respective landmark's three-dimensional position and patch-normal is known.

3.5. Initialization

One of the fundamental difficulties in keyframe-based, monocular SLAM is the inherent chicken-and-egg like nature of the problem: to build a map, the ability to track the camera motion is required, which in turn requires the existence of a map. This issue arises from the fact that no depth information is available: while for stereo-SLAM or RGBD-SLAM the initial map can be built simply from the first image (as the three-dimensional position of features can be estimated from one frame alone) this is not possible for monocular SLAM, as the landmark positions are not fully observable. A common approach to solve this is to apply a separate initialization procedure:

1. Take the first keyframe $\mathcal{K}_1$ and detect promising keypoints $\mathbf{p}_1 \dots \mathbf{p}_n$ using e.g. the FAST corner detector.

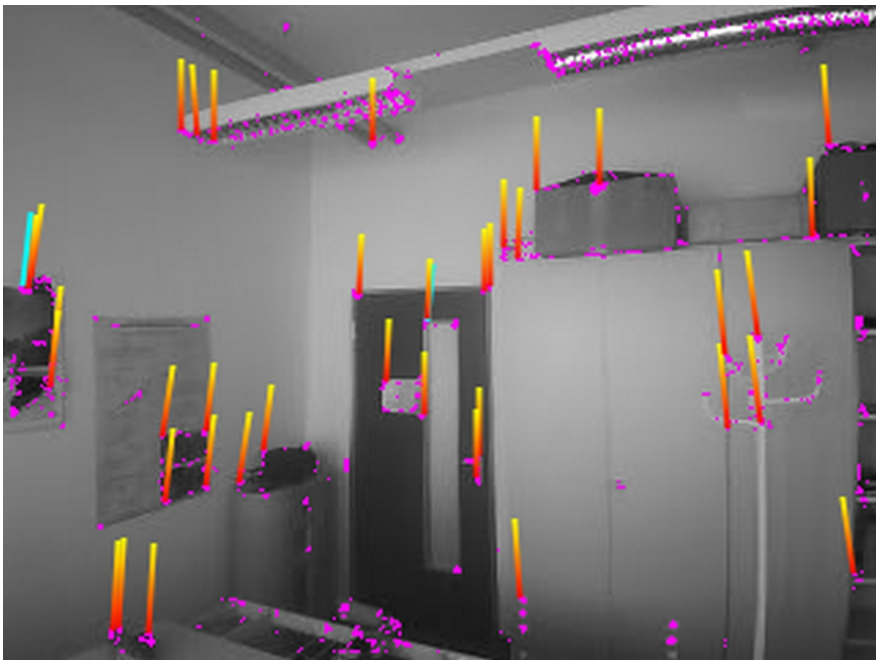


Figure 3.8. Initialization procedure of PTAM. Every line corresponds to a successfully tracked keypoint, going from its location in the first keyframe to its location in the current frame.

2. Track keypoints using a simple frame-to-frame tracking approach as described in Section 3.4, until the camera has moved far enough such that the landmark positions can be triangulated.

3. Take the second keyframe $\mathcal{K}_2$ and extract new keypoint positions $\mathbf{p}'_1 \dots \mathbf{p}'_n$.

4. Generate the initial map from these point-correspondences. The scale of the map as well as the position and orientation of the first keyframe can be defined arbitrarily, due to numerical considerations the initial map is often scaled such that the average distance between camera and landmarks is one.

In the remainder of this section we detail the mathematical methods used for the fourth step, that is: given n corresponding observations $\mathbf{p}_1, \dots, \mathbf{p}_n \in \mathbb{R}^2$ in $\mathcal{K}_1$ and $\mathbf{p}'_1, \dots, \mathbf{p}'_n \in \mathbb{R}^2$ in $\mathcal{K}_2$ and assuming, without loss of generality, that $\mathcal{K}_1 = \mathcal{W}$, the task is to estimate both $\mathbf{E}_{\mathcal{K}_2}$ as well as the landmark positions $\mathbf{x}_1 \dots \mathbf{x}_n$.

This is done by first estimating the **essential matrix E**, encoding the relation between two viewpoints, which we introduce and motivate in Section 3.5.1. We then describe how it can be estimated from n point-correspondences in Section 3.5.2, how the transformation between the two viewpoints is extracted in Section 3.5.3, and how how the landmark positions are triangulated in Section 3.5.4. Finally, in Section 3.5.5, we briefly discuss the accuracy of this method, and how the estimates can be refined using nonlinear optimization. In this section, the transformation to be estimated is denoted by its rotation matrix **R**

and translation vector $\mathbf{t}$:

$$\mathbf{E}_{\mathcal{K}_2} =: \begin{pmatrix} & \mathbf{R} & & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3.5.1. The Essential Matrix

The essential matrix plays a central role in epipolar geometry and encodes the relation between two images, taken of the same scene but from different viewpoints. Note that in this chapter, we focus entirely on the essential matrix $\mathbf{E}$, describing the relationship between points expressed in *homogeneous, normalized image coordinates*: $\tilde{\mathbf{p}}_{\text{norm}} = \mathbf{K}_{\text{cam}}^{-1}\tilde{\mathbf{p}}_{\text{image}}$. However, the same methods can be applied when the two calibration matrices $\mathbf{K}$ and $\mathbf{K}'$ are unknown, using the fundamental matrix $\mathbf{F}$. The essential and the fundamental matrix are related by $\mathbf{E} = \mathbf{K}_{\text{cam}}'^{T}\mathbf{F}\mathbf{K}_{\text{cam}}$. The essential matrix is defined as

$$\mathbf{E} := \mathbf{R}\,[\mathbf{t}]_\times \;\in\; \mathbb{R}^{3\times 3} \tag{3.11}$$

where $[\mathbf{t}]_\times \in \mathbb{R}^{3\times 3}$ is the matrix corresponding to the cross-product with $\mathbf{t}$. Using this definition, it can be deduced that for every pair of corresponding point-observations $\mathbf{p}, \mathbf{p}' \in \mathbb{R}^2$ from two different viewpoints, the relation $\tilde{\mathbf{p}}'^{T}\,\mathbf{E}\,\tilde{\mathbf{p}} = 0$ is satisfied: Let $\mathbf{x} \in \mathbb{R}^3$ be the three-dimensional point in one camera-coordinate system, and $\mathbf{x}' = \mathbf{R}(\mathbf{x} - \mathbf{t})$ the point in the second camera coordinate system. It follows that:

$$(\mathbf{x}')^T\,\mathbf{E}\,\mathbf{x} \overset{(1)}{=} (\mathbf{x} - \mathbf{t})^T\,\mathbf{R}^T\,\mathbf{R}\,[\mathbf{t}]_\times\,\mathbf{x} \overset{(2)}{=} (\mathbf{x} - \mathbf{t})^T\,[\mathbf{t}]_\times\,\mathbf{x} \overset{(3)}{=} \mathbf{0} \tag{3.12}$$

1. substitution of $\mathbf{E} = \mathbf{R}\,[\mathbf{t}]_\times$ and $\mathbf{x}' = \mathbf{R}(\mathbf{x} - \mathbf{t})$
2. as a rotation matrix, $\mathbf{R}$ is orthonormal, hence $\mathbf{R}^T = \mathbf{R}^{-1}$
3. $[\mathbf{t}]_\times\,\mathbf{x} = \mathbf{t} \times \mathbf{x}$ is perpendicular to both $\mathbf{t}$ and $\mathbf{x}$, hence its scalar product with any linear combination of those two vectors is zero

Using $\tilde{\mathbf{p}} = \frac{1}{x_3}\mathbf{x}$ and $\tilde{\mathbf{p}}' = \frac{1}{x_3'}\mathbf{x}'$, equation (3.12) is equivalent to $\tilde{\mathbf{p}}'^{T}\,\mathbf{E}\,\tilde{\mathbf{p}} = 0$.

Not every 3×3 matrix is an essential matrix: As $\mathbf{E}$ is a product of a rotation and a crossproduct matrix, it satisfies a number of properties, in particular every essential matrix has two singular values that are equal while the third one is zero. Interpreted as a projective element, $\mathbf{E}$ has 5 degrees of freedom and is only defined up to scale. This leaves three internal constraints, which can be expressed by $2\mathbf{E}\mathbf{E}^T\mathbf{E} - \text{tr}(\mathbf{E}\mathbf{E}^T)\mathbf{E} = 0$. While this gives nine equations in the elements of $\mathbf{E}$, only three of these are algebraically independent [20].

3.5.2. Estimating the Essential Matrix

As can be seen from the definition (3.11), $\mathbf{E}$ has 6 degrees of freedom. Only 5 of these can however be determined from point correspondences - as mentioned, $\mathbf{E}$ can only be

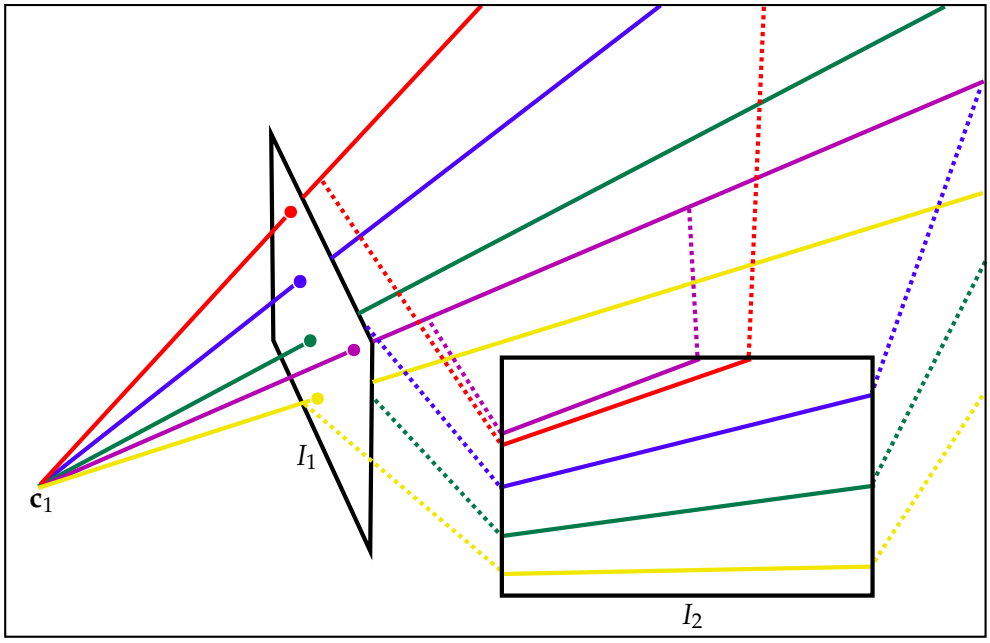


Figure 3.9. Geometric interpretation of the essential matrix: each point-observation $\mathbf{p}$ in I_1 constrains the point to lie on a line in three-dimensional space, passing through the camera center $\mathbf{c}_1$ and $\mathbf{p}$. The projection of this line onto the second image plane I_2 is given by $\mathbf{l} = \mathbf{E}\,\tilde{\mathbf{p}}$. All such lines in I_2 cross in one point, the epipole, which is the projection of $\mathbf{c}_1$.

determined up to scale, leaving the scale of $\mathbf{t}$ unknown. As each pair of observations introduces two additional constraint on $\mathbf{E}$ and one additional unknown, namely the depth of the point, a minimum of 5 points is required to estimate it. In fact there are many different methods to estimate $\mathbf{E}$ from five, six or seven correspondences [20].

A more common approach however is the comparatively simple eight-point algorithm[2], which requires a minimum of 8 point-correspondences: For each pair $\mathbf{p}$, $\mathbf{p}'$, the relationship $\tilde{\mathbf{p}}'^T E \tilde{\mathbf{p}} = 0$ leads to one linear constraint on the entries of $\mathbf{E}$. Combining n such constraints results in a linear system:

$$\mathbf{A}\,\mathbf{e} = \mathbf{0} \tag{3.13}$$

with $\mathbf{A} \in \mathbb{R}^{n\times 9}$ being built by stacking the constraints, while $\mathbf{e} \in \mathbb{R}^9$ contains the entries of $\mathbf{E}$. In the absence of noise and for $n \geq 8$, (3.13) will have a unique solution (up to scale), namely the null-space of $\mathbf{A}$. In the presence of noise and with $n > 8$ however, (3.13) will only yield the trivial solution $\mathbf{e} = \mathbf{0}$. In order to still get an estimate, it is relaxed to

$$\min_{\mathbf{e}} \mathbf{A}\,\mathbf{e} \quad \text{subject to } \|\mathbf{e}\| = 1 \tag{3.14}$$

[2] In OpenCV, the eighth-point algorithm and the seven-point algorithm are implemented. The seven-point algorithm is similar to the eight-point algorithm but exploits the fact that $\det(\mathbf{E}) = 0$, leading to a cubic polynomial which can be solved analytically.

This minimization problem can be solved using the singular value decomposition (SVD) of $\mathbf{A}$, the solution being the right singular vector of $\mathbf{A}$ corresponding to the smallest singular value. Note that this procedure yields a matrix $\mathbf{E} \in \mathbb{R}^{3\times3}$ which is the least-square solution to (3.12). Due to the presence of noise however, this matrix in general does not satisfy the properties of an essential matrix as described above and therefore additional normalization steps are required. Furthermore is has been shown that other methods, exploiting the internal constraints of an essential matrix and solving analytically for the five unknown degrees of freedom are both faster and more stable in practice, however nonlinear and therefore significantly more complex [36].

3.5.3. Estimating Camera-Rotation and Translation

Once $\mathbf{E}$ has been estimated, it is possible to recover $\mathbf{R}$ and $\mathbf{t}$ from it: Let $\mathbf{E} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ be the SVD of $\mathbf{E}$. Algebraic resolution leads to four candidate solutions satisfying (3.11):

$$\begin{aligned} [\mathbf{t}]_\times &= \pm\, \mathbf{V}\,\mathbf{W}\,\mathbf{\Sigma}\,\mathbf{V}^T \\ \mathbf{R} &= \mathbf{U}\,\mathbf{W}^{-1}\,\mathbf{V}^T \end{aligned} \tag{3.15}$$

$$\text{using } \mathbf{W} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{or} \quad \mathbf{W} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

It can be shown that only one of those four solutions is valid, as for any pair of corresponding observations $\mathbf{p}$, $\mathbf{p}'$, three solutions will generate a point $\mathbf{x}$ lying behind at least one camera plane. Furthermore, in practice $[\mathbf{t}]_\times$ will not be a skew-symmetric matrix due to a noisy estimate of $\mathbf{E}$ - in order to avoid this, w_{33} can be set to zero, forcing $[\mathbf{t}]_\times$ to be skew-symmetric. For a derivation of these formulae and further details we refer to the excellent book Multiple View Geometry by Hartley and Zisserman [10].

3.5.4. Triangulating Landmarks

With $\mathbf{R}$ and $\mathbf{t}$ known, the 3D-position of a landmark $\mathbf{x}$ can be triangulated. As each observation yields two constraints on $\mathbf{x}$, this is an over constrained problem. The maximum-likelihood solution $\mathbf{x}^*$ is given by the minimizer of the two-sided reprojection error:

$$\begin{aligned} \mathbf{x}^* &= \arg\min_{\mathbf{x}} E_{\text{rep}}(\mathbf{x}, \mathbf{R}, \mathbf{t}) \quad \text{with} \\ E_{\text{rep}}(\mathbf{x}, \mathbf{R}, \mathbf{t}) &:= \|\text{proj}(\mathbf{K}_{\text{cam}}\mathbf{x}) - \bar{\mathbf{p}}\|^2 + \|\text{proj}(\mathbf{K'}_{\text{cam}}\mathbf{R}(\mathbf{x} - \mathbf{t})) - \bar{\mathbf{p}}'\|^2 \end{aligned} \tag{3.16}$$

where $\bar{\mathbf{p}}$ and $\bar{\mathbf{p}}'$ are the pixel-coordinates of the landmark observations. An analytic solution from any three constraints can be used as initialization, such a solution however is very ill-conditioned and may lead to very poor estimates.

3.5.5. Nonlinear Refinement

Although the eight-point algorithm can easily incorporate more than 8 points, it turns out that doing so does not significantly increase the accuracy of the estimate for $\mathbf{R}$ and $\mathbf{t}$. This

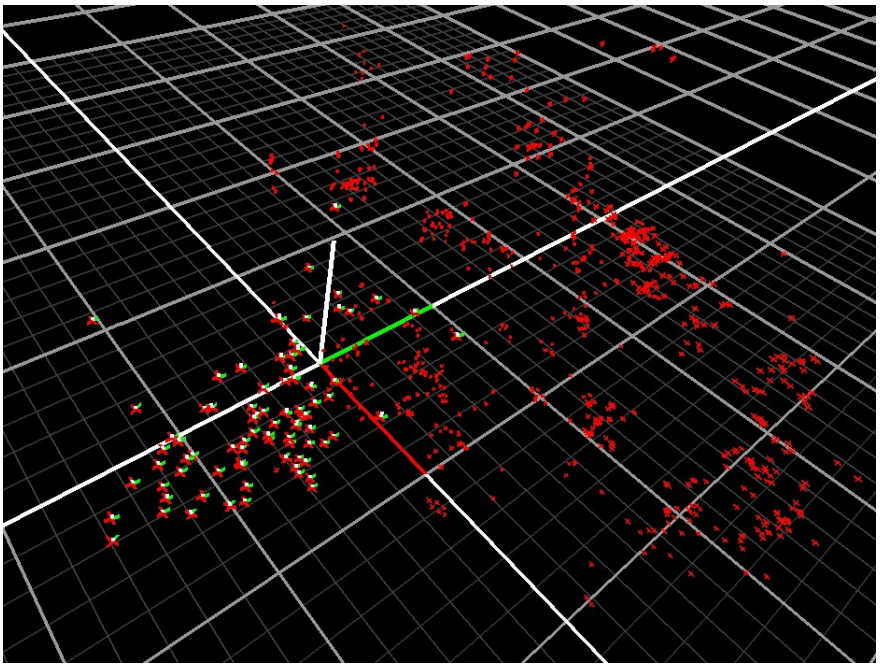


Figure 3.10. The map maintained by PTAM. Keyframes are depicted as red-white-green coordinate frames, landmarks as red crosses.

is due to the fact that it minimizes the algebraic error function (3.13), which does not have a direct geometric interpretation: while without noise this error function does have its minimum at the correct position, in the presence of noise the minimum of this error function does not correspond to the optimal solution in a maximum-likelihood sense. This is particularly true if the point-correspondences are in pixel-coordinates, and (3.13) is solved for the fundamental matrix.

In order to get a better estimate, a nonlinear minimization using the result of the method described above as initialization is applied, minimizing the total reprojection error and solving for $\mathbf{R}$, $\mathbf{t}$ and $\mathbf{x}_1 \dots \mathbf{x}_n$ simultaneously:

$$\min_{\mathbf{x}_1 \dots \mathbf{x}_n, \mathbf{R}, \mathbf{t}} \sum_{i=1}^{n} E_{\text{rep}}(\mathbf{x}_i, \mathbf{R}, \mathbf{t}) \tag{3.17}$$

with $E_{\text{rep}}(\mathbf{x}_i, \mathbf{R}, \mathbf{t})$ defined as in (3.16). This nonlinear refinement is equivalent to a bundle adjustment step as explained in the following Section 3.6.

3.6. Mapping

The mapping loop continuously optimizes the map and extends it by incorporating new keyframes and landmarks.

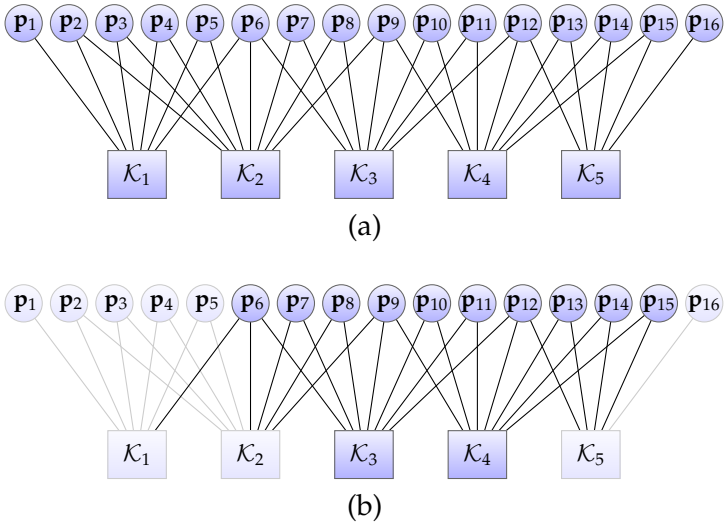


Figure 3.11. Visualization of global (a) versus local (b) BA: each edge corresponds to an observation and respective error term $\mathbf{e}_{ij}$. For the local BA, fixed landmarks and keyframes are drawn shaded, resulting in a large portion of the error terms being constant and hence negligible.

3.6.1. Map Optimization

Given all observations $\mathbf{p}_{ij}$, the goal is to refine the keyframe and landmark positions $\mathcal{K}_j$ and $\mathbf{x}_i$, such that they best coincide with these observations. Assuming independent, Gaussian noise on each observation in pixel, the maximum-likelihood solution to this problem is obtained by minimizing the **total reprojection error** E_{rep}: Let the reprojection error of a single observation $\mathbf{p}$ of landmark $\mathbf{x}_{\mathcal{W}}$, and from a camera-position $\mathcal{C}$ be defined as

$$e(\mathbf{p}, \mathbf{x}_{\mathcal{W}}, \mathcal{C}) := \tilde{\mathbf{p}} - \text{proj}(\mathbf{K}_{\text{cam}} \text{proj}(\mathbf{E}_{\mathcal{C}}\, \tilde{\mathbf{x}}_{\mathcal{W}})) \;\in\; \mathbb{R}^2 \tag{3.18}$$

which corresponds to the distance in pixel between the location where the landmark actually was observed and its projection onto the image. In the remainder of this section, we use $\mathbf{e}_{ij} := e(\mathbf{p}_{ij}, \mathbf{x}_i, \mathcal{K}_j)$ for the reprojection error of an observation $\mathbf{p}_{ij}$ contained in the map. The total reprojection error is now given by:

$$E_{\text{rep}}(\mathbf{x}_1 \dots \mathbf{x}_n, \mathcal{K}_1 \dots \mathcal{K}_m) := \sum_{\substack{j=1\dots m \\ i \in \mathfrak{L}_j}} \text{Obj}\left(\frac{\|\mathbf{e}_{ij}\|^2}{\sigma_{ij}^2} \right) \tag{3.19}$$

where $\text{Obj}\colon \mathbb{R} \to \mathbb{R}$ is a robust kernel function and $\mathfrak{L}_j$ the set of indices of all landmarks observed in keyframe j. Minimizing this error function, using an iterative method such as Levenberg-Marquardt is referred to as global bundle adjustment (BA).

For a growing number of landmarks and keyframes, optimizing this error function as a whole each time a new keyframe or landmark is added quickly becomes computationally unfeasible. This gives rise to the concept of local bundle adjustments. Optimization of (3.19) is performed by only considering a small subset of keyframes and a corresponding

subset of landmarks, keeping everything else fixed: after adding a new keyframe, optimizing only over the most recently added keyframes and a corresponding set of landmarks may be sufficient, assuming that the estimates for older keyframes and landmarks already are comparatively accurate. This is visualized in Figure 3.11. It is to mention, that the total reprojection error (3.19) is a non-convex function and hence only a local and possibly suboptimal minimum will be found.

3.6.2. Adding Keyframes and Landmarks

When a new keyframe has been identified by the tracking part, it can simply be added to the set of keyframes using the known camera position and observations of existing landmarks. To generate new landmarks, additional points from the new frame are extracted and matched to points in other keyframes, using epipolar search - if a match is found, the respective landmark position can be triangulated and is added to the map. Afterwards, a local bundle adjustment step, including the new keyframe and all new landmarks is performed.

3.6.3. Further Mapping Tasks

Apart from bundle adjustments and incorporation of new keyframes, additional routines are required to remove outliers, merge landmarks corresponding to the same three-dimensional point, or identify additional observations and landmarks. The mapping loop continuously executes these tasks (adding new keyframes, performing local BA, performing global BA, pruning the map), ordered by their priority.

3.7. Tracking

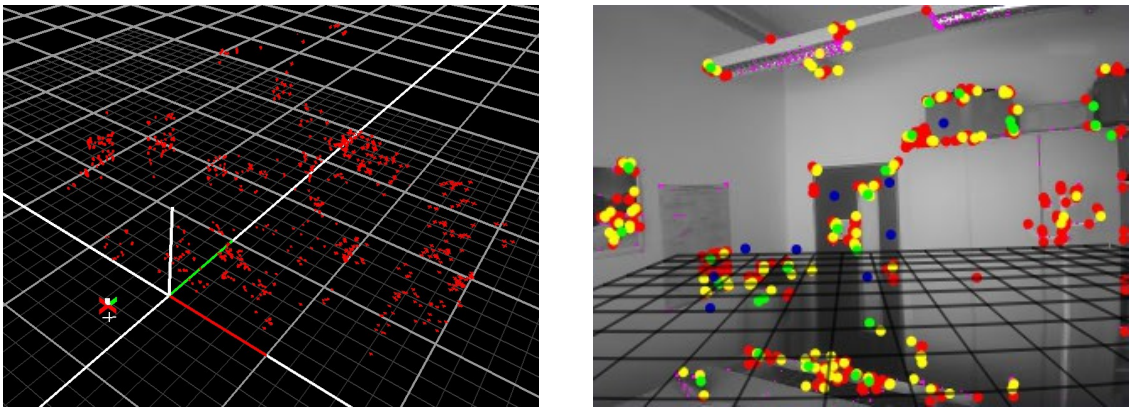
The tracking loop is executed once for each new video-frame I, and calculates the corresponding camera position $\mathcal{C}$, based on the known landmark positions $\mathbf{x}_1 \dots \mathbf{x}_n$. It requires an initial guess of the camera pose $\mathcal{C}_0$, for example the pose in the previous frame.

3.7.1. Pose Estimation

First, all potentially visible landmarks are projected into the image based on the expected camera position $\mathcal{C}_0$, and for each such landmark, a warped template of its expected appearance is generated from a keyframe it was observed in. Using a tracking approach such as described in Section 3.4.3, the exact location of the landmark in the image is then computed to subpixel accuracy. The result of this stage is a set of k 3D-to-2D point correspondences, $\mathbf{x}_1 \dots \mathbf{x}_k$ and $\mathbf{p}_1 \dots \mathbf{p}_k$.

Based on these 3D-to-2D point correspondences, the camera position $\mathcal{C}$ is to be estimated. This is called the perspective n-point (PnP) problem, and is fundamental for many tasks in computer vision and robotics. Again, there are many ways to solve this problem, including iterative and non-iterative methods, a good overview is given in [19].

For a SLAM system, it can generally be assumed that a good initialization is available as the camera movement is small in between two frames - applying a gradient-descend based,



(a) (b)

Figure 3.12. (a) PTAM tracking the camera position $\mathcal{C}$, represented by a red-white-green coordinate frame; landmarks are drawn as red crosses. (b) The respective camera image, the xy-plane of the world-coordinate system is projected into the image as black grid.

iterative method, minimizing the reprojection error as defined in Section 3.6 is therefore the preferred method:

$$\mathcal{C}^* = \arg\min_{\mathcal{C}} \sum_{i=1}^{k} \text{Obj}\left(\frac{\|e(\mathbf{p}_i, \mathbf{x}_i, \mathcal{C})\|^2}{\sigma_i^2}\right) \tag{3.20}$$

Note that the tracking process as described here does not require any feature descriptors (SIFT, SURF, ORB). This is in contrast to many other approaches, in particular filtering-based ones where feature points are frequently matched based on such descriptors.

3.7.2. Tracking Recovery

When tracking is lost, the above method cannot be applied as no initial estimate of the camera position $\mathcal{C}_0$ is available. A separate recovery procedure is therefore required. There are several possibilities to achieve this, for example using viewpoint invariant feature descriptors such as SIFT and SURF. PTAM tries to recover by comparing the downscaled current frame with all existing keyframes, trying to find an approximate match. Based on this match, an initial guess of the camera orientation, assuming the position to be that of the keyframe is computed. The resulting camera pose is then used as $\mathcal{C}_0$ and the normal tracking procedure as described in Section 3.7.1 is applied.

3.7.3. Identifying New Keyframes

The tracking part also decides if a frame will be added as new keyframe based on heuristic criteria such as:

- tracking quality is good (a high fraction of landmarks has been found),
- no keyframe was added for some time,

- no keyframe was taken from a point close to the current camera position.

3.7.4. Further Tracking Aspects

In practice, several improvements are possible. For example, a motion-model can be applied to compute a better initial estimate of the camera position C_0, or an additional coarse tracking stage, only considering keypoints found on a higher scale may be added in order to track fast motion.

3.8. Summary

In this chapter, we first gave an overview over the monocular SLAM problem, how it can be solved and reviewed the current state of the art. We distinguished between two distinct approaches - the filtering-based and the keyframe-based approach - and identified their similarities and differences. We introduced keypoints as basic building block of both approaches, and described in detail how they can be defined, found and tracked in an image. Furthermore, we identified the three main components of keyframe-based, monocular SLAM algorithms (initialization, tracking and mapping), and how they interact. We proceeded to detailing these three components and the fundamental mathematical methods applied.

4. Data Fusion and Filtering

In robotics one generally deals with dynamic systems, i.e. (physical) systems that change their state over time. For a quadrocopter for example, the state might consist of its current position, orientation and velocity. In order to accurately control such a system, sensors are used to collect information and infer the current state as accurately as possible. Measurements from real-world sensors however are always subject to measurement errors (also called noise), hence using only the most recent measurement often leads to unstable and poor results. Furthermore in many cases multiple sensors can be used to acquire information about the same state variable - in order to get an optimal estimate, these measurements can then be combined to give a single estimate with higher accuracy. The goal of data fusion and filtering algorithms is to use sensor measurements and knowledge about the dynamics of the system to gain an accurate estimate of the system's *current* state.

In this chapter, we introduce the linear Kalman filter in Section 4.1 and its extension to nonlinear systems, the extended Kalman filter, in Section 4.2. In Sections 4.3 and 4.4 we briefly introduce two further filtering methods, the unscented Kalman filter and particle filters.

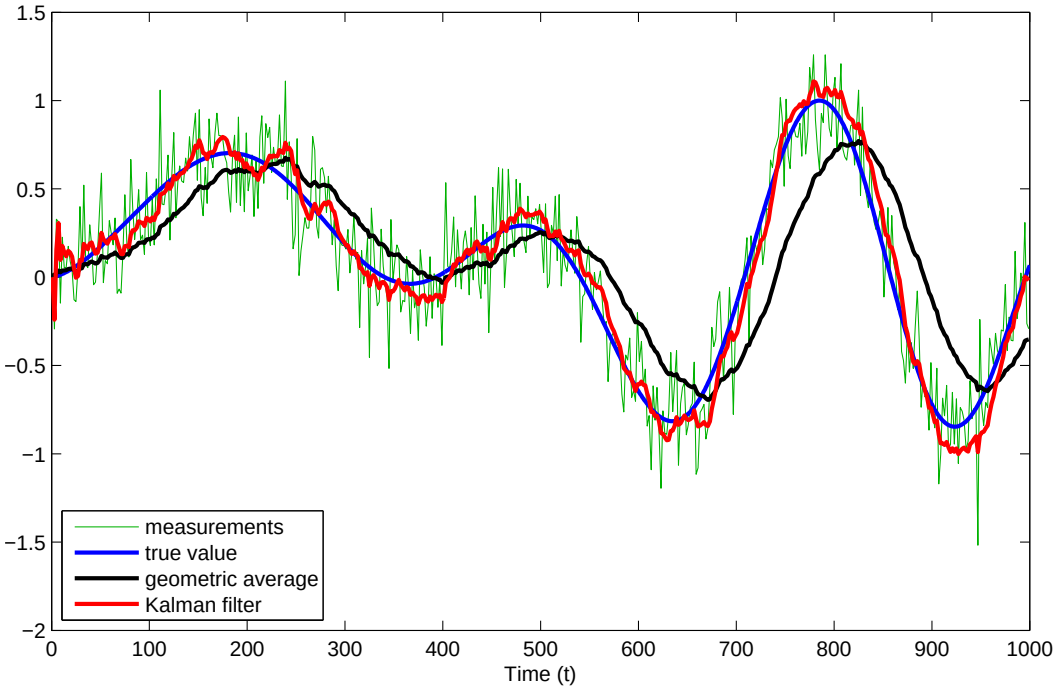


Figure 4.1. Example of filtering: the true state is shown in blue, the (noisy) measurements x_t are shown in green. The black line corresponds to a simple geometric-decay filter - while this does lead to significantly smoother results, an artificial delay is introduced. The red line shows the result of a Kalman filter estimating x and $\dot{x}$.

4.1. The Linear Kalman Filter

The Kalman filter is a well-known method to filter and fuse noisy measurements of a dynamic system to get a good estimate of the current state. It assume,s that all observed and latent variables have a (multivariate) Gaussian distribution, the measurements are subject to independent, Gaussian noise and the system is linear. Under these assumptions it can even be shown that the Kalman filter is the optimal method to compute an estimate of the current state as well as the uncertainty (covariance) of this estimate. In the remainder of this section, we use the following notation:

- n, m, d: dimension of the state vector, measurement vector and control vector respectively,
- $\mathbf{x}_k \in \mathbb{R}^n$: true state at time k. The estimate of this vector, incorporating measurements up to and including the measurement at time j is denoted by $\hat{\mathbf{x}}_{k|j}$,
- $\mathbf{P}_{k|j} \in \mathbb{R}^{n \times n}$: estimated covariance of $\hat{\mathbf{x}}_{k|j}$,
- $\mathbf{B} \in \mathbb{R}^{n \times d}$: control-input model, mapping the control vector $\mathbf{u}_k \in \mathbb{R}^d$ to its effect on the internal state,
- $\mathbf{F} \in \mathbb{R}^{n \times n}$: state transition model, mapping the state at time $k-1$ to the state at time k. This transition is assumed to be subject to zero-mean Gaussian noise $\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q})$, where $\mathbf{Q}$ is known: $\mathbf{x}_k = \mathbf{F}\,\mathbf{x}_{k-1} + \mathbf{B}\,\mathbf{u}_k + \mathbf{w}_k$.
- $\mathbf{z}_k \in \mathbb{R}^m$: observation at time k. Again this observation is assumed to be subject to zero-mean Gaussian noise $\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R})$, where $\mathbf{R}$ is known: $\mathbf{z}_k = \mathbf{H}\,\mathbf{x}_k + \mathbf{v}_k$, where $\mathbf{H} \in \mathbb{R}^{m \times n}$ is the observation model, mapping the internal state to the respective expected observation.

Using the above definitions, the linear Kalman filter now operates in a continuous prediction-update-loop:

1. **predicting** the state $\mathbf{x}$ ahead in time, increasing uncertainty:

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= \mathbf{F}\,\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}\,\mathbf{u}_k \\ \mathbf{P}_{k|k-1} &= \mathbf{F}\,\mathbf{P}_{k-1|k-1}\,\mathbf{F}^T + \mathbf{Q} \end{aligned} \tag{4.1}$$

2. **updating** the state $\mathbf{x}$ by incorporating an observation, decreasing uncertainty:

$$\begin{aligned} \mathbf{y}_k &= \mathbf{z}_k - \mathbf{H}\,\hat{\mathbf{x}}_{k|k-1} \\ \mathbf{S}_k &= \mathbf{H}\,\mathbf{P}_{k|k-1}\,\mathbf{H}^T + \mathbf{R} \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1}\,\mathbf{H}^T\,\mathbf{S}_k^{-1} \\ \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\,\mathbf{y}_k \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k\,\mathbf{H})\,\mathbf{P}_{k|k-1} \end{aligned} \tag{4.2}$$

A full derivation and proof of these formulae is beyond the scope of this work, for further details we refer to the excellent book Probabilistic Robotics by Thrun et al. [40].

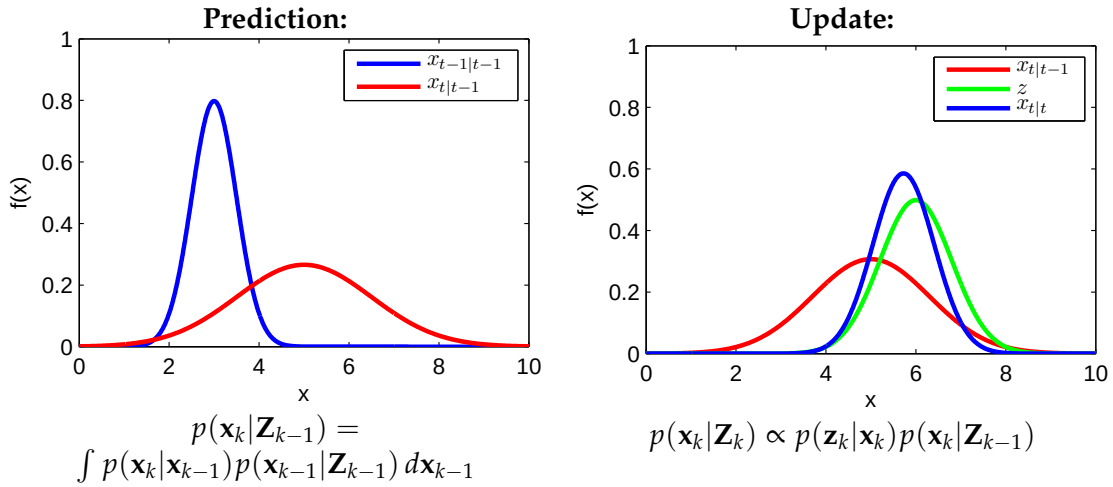


Figure 4.2. Prediction and update step for a one-dimensional Kalman filter, with $\mathbf{Z}_k := \{\mathbf{z}_1 \dots \mathbf{z}_k\}$. While the prediction step corresponds to a convolution of two Gaussians, the update step is an application of Bayes formula: *posterior* $\propto$ *likelihood* $\cdot$ *prior*. For both steps, the resulting distribution again is an analytically computable Gaussian.

4.2. The Extended Kalman Filter

The extended Kalman filter drops the assumption of a linear system, making it applicable to a much wider range of real-world problems. The only difference is that observation, state transition and control model can now be defined by any two differentiable functions $f\colon \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^n$ and $h\colon \mathbb{R}^n \to \mathbb{R}^m$:

$$\begin{aligned} \mathbf{x}_k &= f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1} \\ \mathbf{z}_k &= h(\mathbf{x}_k) + \mathbf{v}_k \end{aligned} \tag{4.3}$$

The difficulty now lies in the fact that when we apply a nonlinear transformation to a Gaussian random variable, the resulting random variable is no longer Gaussian: in order to still make the above framework applicable, h and f are approximated by a first-order Taylor approximation, which however leads to the result no longer being optimal. Let

$$\begin{aligned} \mathbf{F}_{k-1} &:= \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k} \\ \mathbf{H}_k &:= \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k|k-1}} \end{aligned} \tag{4.4}$$

Update and prediction can then be approximated as follows, the only differences to the linear Kalman filter in addition to $\mathbf{H}$ and $\mathbf{F}$ now being different for every step are highlighted in red:

1. **prediction**:

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \\ \mathbf{P}_{k|k-1} &= \mathbf{F}_{k-1}\,\mathbf{P}_{k-1|k-1}\,\mathbf{F}_{k-1}^T + \mathbf{Q} \end{aligned} \tag{4.5}$$

2. **update**:

$$\begin{aligned} \mathbf{y}_k &= \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}) \\ \mathbf{S}_k &= \mathbf{H}_k\,\mathbf{P}_{k|k-1}\,\mathbf{H}_k^T + \mathbf{R} \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1}\,\mathbf{H}_k^T\,\mathbf{S}_k^{-1} \\ \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\,\mathbf{y}_k \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k\,\mathbf{H}_k)\,\mathbf{P}_{k|k-1} \end{aligned} \tag{4.6}$$

4.3. The Unscented Kalman Filter

The unscented Kalman filter is a further improvement on the extended Kalman filter, often leading to more robust and accurate results [12]. It addresses the problem that using a first-order Taylor approximation to transform a probability distribution in general leads to poor results when f is highly nonlinear: instead of linearization, the transformed distribution is estimated from a number of nonlinearly transformed sample points from the original distribution.

4.4. Particle Filters

Particle filters further relax the assumptions made: both state and observation are no longer required to have a Gaussian distribution - allowing for example to track multiple hypothesis simultaneously. This is achieved by characterizing the distribution using a set of sample points called particles. The major drawback of this method is, that the number of particles required grows exponentially in the dimension of the state - often rendering this approach computationally unfeasible for large n.

5. Control

Control theory deals with the problem of controlling the behavior of a dynamic system, i.e. a (physical) system that changes its state over time and which can be controlled by one or more system input values. The general goal is to calculate system input values $\mathbf{u}(t)$, such that the system reaches and holds a desired state. In other words, the measured error $\mathbf{e}(t)$ between a given setpoint $\mathbf{w}(t)$ and the measured output of the system $\mathbf{y}(t)$ is to be minimized over time. In particular, the goal is to quickly reach the desired setpoint and hold it without oscillating around it, counteracting any random disturbances introduced into the system by the environment. This process is schematically represented in Figure 5.1.

In this chapter, we present the proportional-integral-derivative controller (PID controller), a generic control loop feedback mechanism widely used in industrial control systems. A PID controller is used in our approach to directly control the quadrocopter. It is based on three separate control mechanisms, the control signal being a weighted sum of all three terms:

- the **proportional** part depends on the current error $\mathbf{e}(t)$,
- the **integral** part depends on the accumulated past error $\int_0^t \mathbf{e}(\tau)d\tau$,
- the **derivative** part depends on the predicted future error, based on the derivative of the error with respect to time $\dot{\mathbf{e}}(t)$.

If integral and derivative of the error cannot be measured directly, they are approximated by numeric integration and differentiation: $\int_0^t \mathbf{e}(\tau)d\tau \approx \sum_{\tau=1}^{t} \mathbf{e}(\tau)$ and $\dot{\mathbf{e}}(t) \approx \frac{1}{\delta_t}(\mathbf{e}(t) - \mathbf{e}(t-\delta_t))$. The PID controller now calculates the system input values according to

$$\mathbf{u}(t) = K_p\,\mathbf{e}(t) + K_i \int_0^t \mathbf{e}(\tau)d\tau + K_d\,\dot{\mathbf{e}}(t) \tag{5.1}$$

where K_p, K_i and K_d are tunable parameters that typically are determined experimentally by the means of trial-and-error, however there are heuristic methods and guidelines to

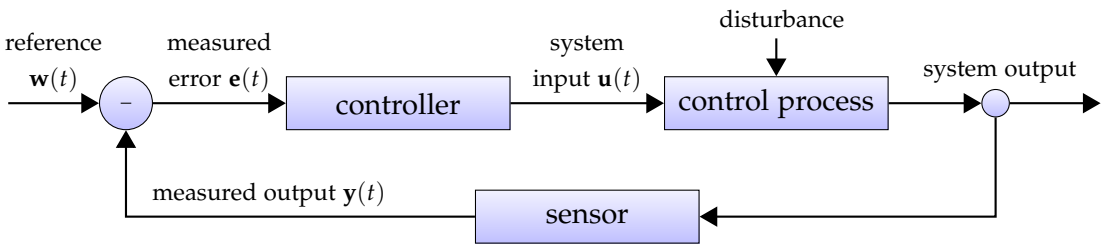


Figure 5.1. Schematic representation of a general control loop. The goal is to calculate system input values $\mathbf{u}(t)$ such that the measured error $\mathbf{e}(t) = \mathbf{w}(t) - \mathbf{y}(t)$ is minimized.

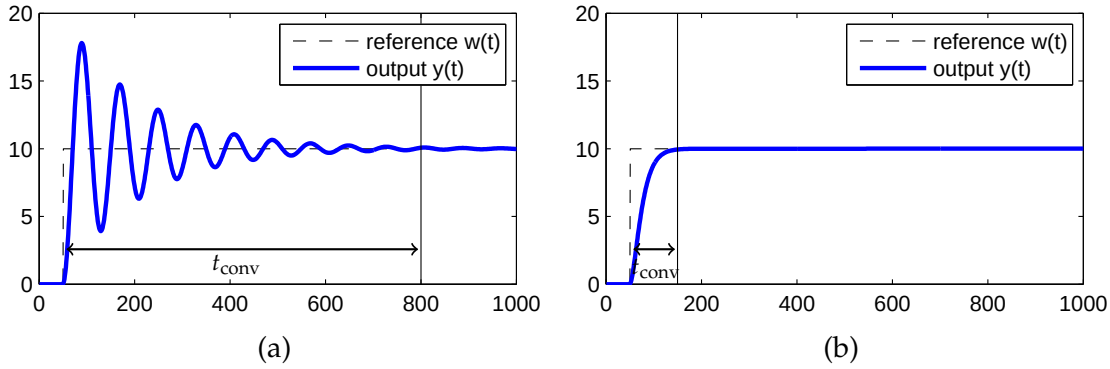


Figure 5.2. (a) proportional control. (b) the same system when adding a derivative term.

help this process. The quality of a control system can be measured by the convergence time t_{conv}, measuring how long it takes until $\mathbf{e}(t)$ stays within an specified, small interval around zero. The effect of these three parts of a PID-controller is explained below.

The Proportional Term

The proportional part is always required, and is the part responsible for reducing the error: the bigger the error, the stronger the control signal. In real-world systems however, a purely proportional controller causes severe overshoot, leading to strong oscillations. The behavior of a typical system[1], controlled by a purely proportional controller is visualized in Figure 5.2a.

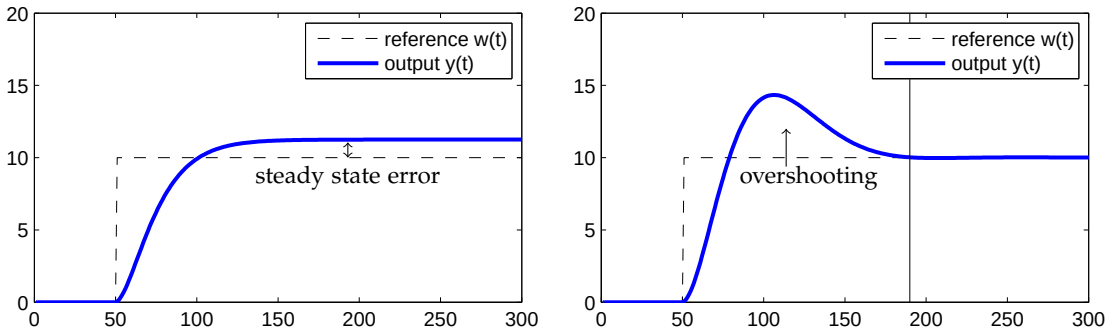
The Derivative Term

The derivative part has the effect of dampening occurring oscillations: the higher the rate of change of the error, the more this term contributes towards slowing down this rate of change, reducing overshoot and oscillations. The effect of adding a derivative term to the controller is visualized in Figure 5.2b.

The Integral Term

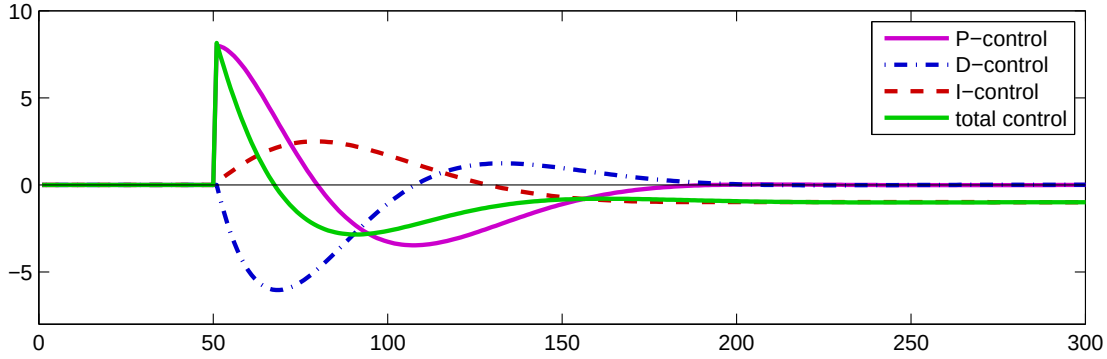
The integral part is responsible for eliminating steady-state errors: for a biased system requiring a constant control input to hold a state, a pure PD-controller will settle above or below the setpoint. Depending on accumulated past error, the integral term compensates for this bias - it however needs to be treated with caution as it may increases convergence time and cause strong oscillations. The effect of adding a derivative term to a PD-controller is visualized in Figure 5.3.

[1] a first-order time delay system



(a) PD-control

(b) PID-control



(c) The three control components, corresponding to the state as plotted in (b)

Figure 5.3. (a) PD-control of a biased system, resulting in a steady-state error. (b) the same system when adding an integral term. While for this system overshooting is unavoidable, the integral term causes it to still settle at the correct value. (c) the three distinct control terms working together.

6. Scale Estimation for Monocular SLAM

One of the major shortcomings of monocular SLAM methods is, that the scale $\lambda \in \mathbb{R}$ of the map cannot be determined without using either additional sensor information or knowledge about objects present in the scene. To use a monocular SLAM method for navigation however, an estimate for this scale factor is essential. Online estimation of this scaling factor by combining visual keypoints and inertial measurements (acceleration) has recently been addressed in the literature [30, 15] and used in practice [1]. In this chapter, we analyze the problem from a statistical perspective and derive a novel method to estimate the scaling factor particularly suited for the sensors available on the Parrot AR.Drone.

6.1. Problem Formulation and Analysis

The goal is to accurately and online estimate the scale of a monocular SLAM system from a continuous sequence of noisy pose estimates $\mathbf{p}_v(t)$ computed by this SLAM system, and a sequence of noisy sensor measurements of the absolute position $\mathbf{p}_s(t)$, speed $\mathbf{v}_s(t)$ or acceleration $\mathbf{a}_s(t)$. We derive a method for the case when $\mathbf{v}_s(t)$ can be measured drift-free, as it is the case for the Parrot AR.Drone: The horizontal speeds are calculated onboard by the drone, using the ultrasound altimeter and an optical-flow based algorithm on the floor camera. The ultrasound altimeter measures the relative height, which is subject to "drift" for uneven ground.

In regular time intervals, the distance traveled within the last interval is measured by both the SLAM system and the available sensors, creating a pair of measurements $\mathbf{x}_i, \mathbf{y}_i$:

$$\begin{aligned} \mathbf{x}_i &:= \mathbf{p}_v(t_i) - \mathbf{p}_v(t_{i-1}) \\ \mathbf{y}_i &:= \int_{t_{i-1}}^{t_i} \mathbf{v}_s(t)dt \end{aligned} \tag{6.1}$$

Assuming independent Gaussian noise on all measurements, the noise on $\mathbf{x}_i$ and $\mathbf{y}_i$ is also independent and Gaussian. Furthermore we assume that

- the variance of the noise on $\mathbf{x}_i$ is a constant σ_x^2, and independent of λ: The noise on $\mathbf{p}_v(t)$ depends on many factors, but in particular it scales linearly with the depth of the feature points observed. As the initial scale of the map can be chosen such that the average feature depth is constant, the noise on the pose estimate scaled according to the SLAM map is constant assuming the average feature depth not to change significantly over time. At this point we do not consider other influences on the pose estimation accuracy such as the number of visible landmarks.

- the variance of the noise on $\mathbf{y}_i$ is a constant σ_y^2, which follows directly from the noise on $\mathbf{v}_s(t)$ having a constant variance. σ_y^2 does however scale linearly with the interval size.

This leads to the following statistical problem formulation: Given two sets of corresponding, noisy, d-dimensional measurements $X := \{\mathbf{x}_1 \dots \mathbf{x}_n\}$ and $Y := \{\mathbf{y}_1 \dots \mathbf{y}_n\}$ of the true distance covered, the task is to estimate the unknown scaling factor λ. The samples are assumed to be distributed according to

$$\begin{aligned} \mathbf{x}_i &\sim \mathcal{N}(\lambda \boldsymbol{\mu}_i, \sigma_x^2) \\ \mathbf{y}_i &\sim \mathcal{N}(\boldsymbol{\mu}_i, \sigma_y^2) \end{aligned} \tag{6.2}$$

where $\boldsymbol{\mu}_i \in \mathbb{R}^d$ denotes the true (unknown) distance covered and $\sigma_x^2, \sigma_y^2 \in \mathbb{R}^+$ are the known variances of the measurement errors.

6.2. Derivation of the ML Estimator for the Scale

Following a maximum likelihood (ML) approach, the negative log-likelihood of obtaining the measurements X, Y is given by:

$$\mathcal{L}(\boldsymbol{\mu}_1 \dots \boldsymbol{\mu}_n, \lambda) \propto \frac{1}{2} \sum_{i=1}^{n} \left(\frac{\|\mathbf{x}_i - \lambda \boldsymbol{\mu}_i\|^2}{\sigma_x^2} + \frac{\|\mathbf{y}_i - \boldsymbol{\mu}_i\|^2}{\sigma_y^2} \right) \tag{6.3}$$

The ML estimator for λ can now be calculated as

$$\lambda^* := \arg\min_{\lambda} \min_{\boldsymbol{\mu}_1 \dots \boldsymbol{\mu}_n} \mathcal{L}(\boldsymbol{\mu}_1 \dots \boldsymbol{\mu}_n, \lambda) \tag{6.4}$$

Setting the derivative of $\mathcal{L}(\boldsymbol{\mu}_1 \dots \boldsymbol{\mu}_n, \lambda)$ with respect to $\boldsymbol{\mu}_i$ to zero, we can calculate the optimal values for $\boldsymbol{\mu}_i$ in terms of λ:

$$\begin{aligned} \frac{\partial \mathcal{L}(\boldsymbol{\mu}_1 \dots \boldsymbol{\mu}_n, \lambda)}{\partial \boldsymbol{\mu}_i} &\propto \frac{\lambda^2 \boldsymbol{\mu}_i - \lambda \mathbf{x}_i}{\sigma_x^2} + \frac{\boldsymbol{\mu}_i - \mathbf{y}_i}{\sigma_y^2} \stackrel{!}{=} \mathbf{0} \\ \Rightarrow \boldsymbol{\mu}_i &= \frac{\lambda \sigma_y^2 \mathbf{x}_i + \sigma_x^2 \mathbf{y}_i}{\lambda^2 \sigma_y^2 + \sigma_x^2} \end{aligned} \tag{6.5}$$

Substituting (6.5) into (6.4) and simplifying gives

$$\begin{aligned} \lambda^* &= \arg\min_{\lambda} \hat{\mathcal{L}}(\lambda) \\ \text{with } \hat{\mathcal{L}}(\lambda) &:= \min_{\boldsymbol{\mu}_1 \dots \boldsymbol{\mu}_n} \mathcal{L}(\boldsymbol{\mu}_1 \dots \boldsymbol{\mu}_n, \lambda) \propto \frac{1}{2} \sum_{i=1}^{n} \frac{\|\mathbf{x}_i - \lambda \mathbf{y}_i\|^2}{\lambda^2 \sigma_y^2 + \sigma_x^2} \end{aligned} \tag{6.6}$$

Setting the derivative with respect to λ to zero leads to

$$\hat{\mathcal{L}}'(\lambda) \propto \sum_{i=1}^{n} \frac{(\lambda \mathbf{y}_i - \mathbf{x}_i)^T (\sigma_x^2 \mathbf{y}_i + \lambda \sigma_y^2 \mathbf{x}_i)}{(\lambda^2 \sigma_y^2 + \sigma_x^2)^2} \stackrel{!}{=} \mathbf{0}$$

which simplifies to a quadratic equation in λ. Using the observations that

- $\lim_{\lambda\to\infty} \hat{\mathcal{L}}(\lambda) = \lim_{\lambda\to-\infty} \hat{\mathcal{L}}(\lambda) \in \mathbb{R}$,
- $\exists\lambda_0 \in \mathbb{R}^+ \colon \forall\lambda > \lambda_0 \colon \hat{\mathcal{L}}'(\lambda) > 0 \wedge \hat{\mathcal{L}}'(-\lambda) > 0$, i.e. for large enough positive or negative λ, $\hat{\mathcal{L}}'(\lambda)$ becomes positive (This only holds if and only if $\sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{y}_i > 0$, which is the case for a reasonable set of measurements.),
- $\hat{\mathcal{L}}(\lambda)$ has at most two extrema,

it can be concluded that $\hat{\mathcal{L}}(\lambda)$ has a unique, global minimum at

$$\begin{gathered} \lambda^* = \frac{s_{xx} - s_{yy} + \sqrt{(s_{xx} - s_{yy})^2 + 4s_{xy}^2}}{2\sigma_x^{-1}\sigma_y s_{xy}} \\ \text{with } s_{xx} := \sigma_y^2 \sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{x}_i \quad s_{yy} := \sigma_x^2 \sum_{i=1}^{n} \mathbf{y}_i^T \mathbf{y}_i \quad s_{xy} := \sigma_y \sigma_x \sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{y}_i \end{gathered} \tag{6.7}$$

which is the ML estimator for the scale factor λ.

6.3. The Effect of Measurement Noise

The above derivation shows, that the variances of the measurement errors σ_x^2 and σ_y^2 have a non-negligible influence on the estimate for λ. The reason for this becomes clear with the following example: We artificially generate measurement pairs $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^3$ according to (6.2) and estimate λ by

1. scaling the $\mathbf{y}_i$ such that they best correspond to the $\mathbf{x}_i$

$$\lambda_y^* := \arg\min_{\lambda} \sum_i \|\mathbf{x}_i - \lambda \mathbf{y}_i\|^2 = \frac{\sum_i \mathbf{x}_i^T \mathbf{y}_i}{\sum_i \mathbf{y}_i^T \mathbf{y}_i} \tag{6.8}$$

2. scaling the $\mathbf{x}_i$ such that they best correspond to the $\mathbf{y}_i$

$$\lambda_x^* := \left(\arg\min_{\lambda} \sum_i \|\lambda \mathbf{x}_i - \mathbf{y}_i\|^2 \right)^{-1} = \frac{\sum_i \mathbf{x}_i^T \mathbf{x}_i}{\sum_i \mathbf{x}_i^T \mathbf{y}_i} \tag{6.9}$$

See Figure 6.1 for an example. It turns out that $\lambda_x^* = \lim_{\sigma_x\to 0} \lambda^*$ and $\lambda_y^* = \lim_{\sigma_y\to 0} \lambda^*$, i.e. the two above solutions correspond to the ML estimator if one of the measurements sources is assumed to be noise-free. In practice however this will never be the case, and none of these two limit cases will yield an unbiased result. In essence this means that in order to estimate λ, the variances of the measurement errors σ_x^2 and σ_y^2 need to be known (or estimated), otherwise only a lower and an upper bound for λ can be given (λ^* always lies in between λ_y^* and λ_x^*).

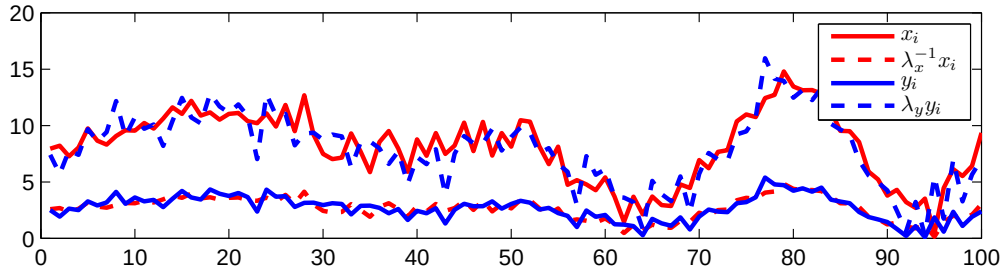


Figure 6.1. 100 artificially generated measurement pairs (x component), with $\lambda = 3$, $\sigma_x^2 = 1$ and $\sigma_y^2 = 0.3$. The dotted lines correspond to the with $\lambda_x^* = 3.05$ and $\lambda_y^* = 2.82$ scaled measurements respectively. When adding more measurement pairs with these particular values, λ_x^* converges to 3.07, while λ_y^* converges to 2.81.

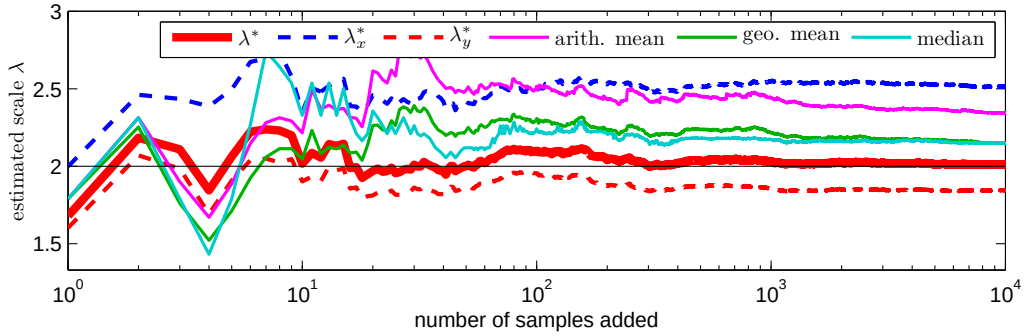


Figure 6.2. Estimated scale for the different estimators as more samples are added ($\lambda = 2$, $\sigma_x = 1$, $\sigma_y = 0.3$), observe that only λ^* converges to the correct value.

6.4. Test with Synthetic Data

In this section, we compare the estimator proposed λ^* with λ_x^*, λ_y^* and three further estimation strategies on synthetically generated data according to the model (6.2), drawing the true distances covered $\boldsymbol{\mu}_i$ from a normal distribution $\mathcal{N}(\mathbf{0}_3, \mathbf{I}_{3\times 3})$:

$$\text{arith. mean} := \frac{1}{n} \sum_i \frac{\|\mathbf{x}_i\|}{\|\mathbf{y}_i\|} \tag{6.10}$$

$$\text{geo. mean} := \left(\prod_i \frac{\|\mathbf{x}_i\|}{\|\mathbf{y}_i\|} \right)^{\frac{1}{n}} \tag{6.11}$$

$$\text{median} := \text{median} \left\{ \frac{\|\mathbf{x}_i\|}{\|\mathbf{y}_i\|}, i = 1 \dots n \right\} \tag{6.12}$$

As expected, all these estimators are biased, i.e. by adding more measurements they converge to a value (significantly) different from the real scaling factor λ, this can be observed in Figure 6.2. Figure 6.3 shows how this bias increases when increasing the measurement noise. In all our experiments (varying the true scale λ, the noise variances σ_x, σ_y, as well as the number of dimensions considered) the proposed method is the only one converging to the true scale factor. The differences become much less significant when the measurement noise is reduced. The influence of wrong σ_x, σ_y is analyzed in Figure 6.4.

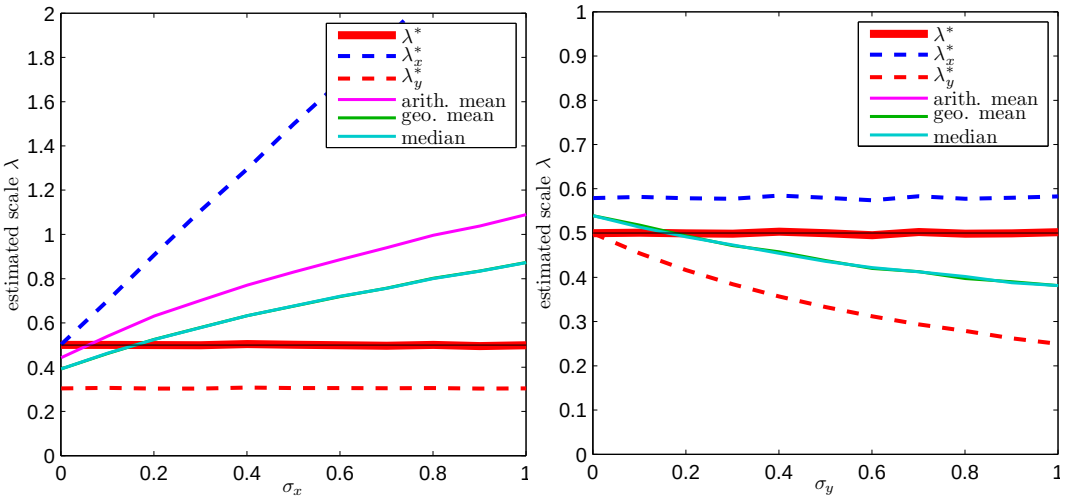


Figure 6.3. Estimated scale for 50,000 samples and different values for (a) σ_x and (b) σ_y: observe how λ_y^* becomes more biased for large σ_y, while the bias of λ_x^* depends only on σ_x.

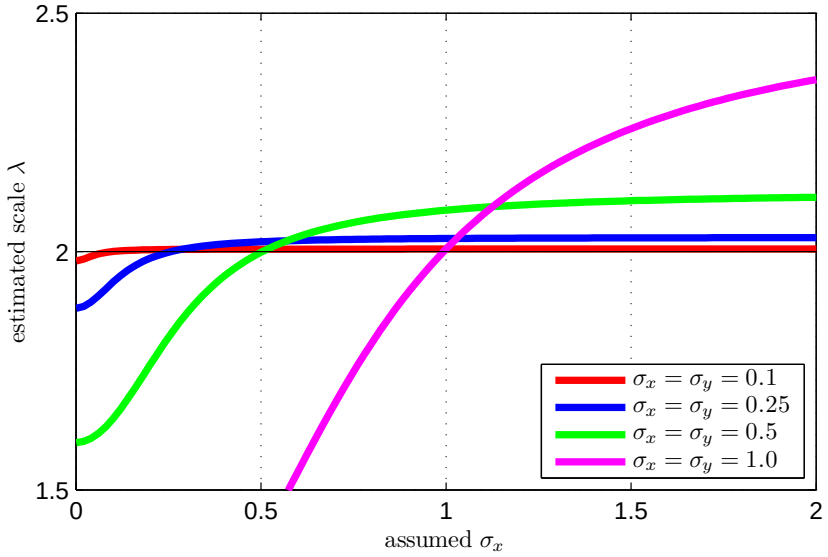


Figure 6.4. Effect of incorrect σ_x: Four sets of samples with different true noise levels and a true scale of $\lambda = 2$ are generated. We then estimate the scale using our approach, with the correct value for σ_y and a range of different values for σ_x: observe how the bias caused by a wrong value for σ_x becomes significantly larger for large noise levels. Note that $\sigma_x = \sigma_y = 1$ corresponds to the measurement error being approximately as large as the true distances moved, while $\sigma_x = \sigma_y = 0.1$ corresponds to the measurement error being approximately 10% of the true distances moved.

6.5. Summary

In this chapter, we derived a novel closed-form method to estimate the scaling factor λ of a monocular SLAM system using for example an IMU and an ultrasound altimeter. In order to deal with drift (IMU) or discontinuities (ultrasound altimeter over uneven ground), the moved path is split into intervals, generating two noisy sets of measurements $\mathbf{X}, \mathbf{Y}$. We argued that, in order to estimate λ from such sets of measurements without bias, the approximate variances of the measurement errors σ_x^2, σ_y^2 need to be known. Our method is computationally inexpensive and flexible: outliers can easily be removed from the set of measurements, and data from multiple sensors can easily be incorporated into the estimation.

In practice, the measurement noise of at least one source will be small and hence rough approximations of σ_x and σ_y lead to accurate results: if the standard deviation of the measurement error is approximately 10% of the true distances moved, the bias for unknown σ_x^2 or σ_y^2 will be less than 2% (see Figure 6.4).

7. Implementation

In this chapter, we describe our approach to the problem and the system developed and implemented in the course of this thesis. In Section 7.1, we first outline our approach, summarizing the methods used and how they interact. We briefly present the software architecture in Section 7.2. In the following three sections, we describe the three main components of our approach: in Section 7.3, we describe the monocular SLAM system, how it is augmented to benefit from the additional sensor measurements available and how the method derived in Chapter 6 is used to estimate the scale of the map. In Section 7.4, we describe the extended Kalman filter used to fuse information from the different sensors available, the prediction model developed and how time delays are compensated for. In Section 7.5, we describe the PID-controller controlling the drone.

7.1. Approach Outline

Our approach consists of three major components:

- **Monocular SLAM:** a monocular SLAM algorithm as described in Chapter 3 (PTAM) is applied to incoming video frames and computes an estimate of the drone's pose, based on a predicted pose calculated by the Kalman filter. By comparing sensor data with PTAM's pose estimate, the initially unknown scale of the map is estimated using our method as described in Chapter 6.

- **Extended Kalman filter:** in order to fuse the pose estimate provided by PTAM with available sensor information as well as the predicted effect of sent control commands, an extended Kalman filter (EKF) as introduced in Chapter 4 is developed. It not only computes a more accurate estimate of the drone's pose and speed at each video frame, but also a prediction of its future state when the next control command will take effect, compensating for delays in the communication process.

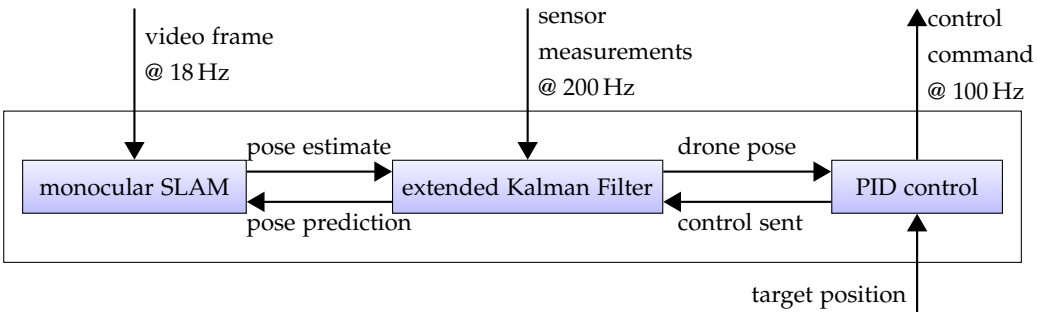


Figure 7.1. Outline of our approach.

- **PID controller:** based on an estimate of the drone's position and speed provided by the EKF, a PID-controller as introduced in Chapter 5 is used to calculate appropriate control commands to fly to and hold a given target position.

7.2. Software Architecture

The first task to be solved for this thesis was to build a reliable and fault-tolerant software system controlling the Parrot AR.Drone. In particular, the challenge was to develop a fault-tolerant system, for example it needs to quickly and automatically detect and restore a lost connection to the drone without loosing the internal state of the control system. We achieved this by encapsulating all SDK-dependent code and direct drone-communication in a separate process: the **drone proxy process** offers a clean local interface for accessing video and navigational data, as well as sending control commands to the drone. Furthermore it can record and replay all data sent by the drone, simulating a real flight and significantly facilitating the debugging process.

A second process, the **UI process** monitors the state of the drone proxy and automatically restarts it when a connection loss or error is detected. It also offers the possibility of manually controlling the drone via keyboard or gamepad and serves as a graphical interface for managing drone connection and recording or replaying flights.

The main algorithmic component runs in a third process: the **control process** contains the SLAM system, the Kalman filter and the PID-controller, as well as video and map visualization. When controlling the drone manually, this process is not required - allowing for immediate manual takeover when it is interrupted for debugging purposes.

These three processes communicate using named pipes for interchanging message, as well as shared memory regions, mutexes and events for asynchronously interchanging video, navigational and control data. The software architecture is visualized in Figure 7.2.

7.3. Monocular SLAM

For the task of monocular SLAM, our solution is based on the open-source parallel tracking and mapping system by Klein and Murray [14], a well-known and widely used keyframe-based monocular SLAM system as described in Chapter 3. It is augmented to make use of the additional sensor information available, in particular the main drawback of monocular SLAM, the impossibility to recover the scale of the scene, is resolved.

7.3.1. Scale Estimation

In order to navigate a quadrocopter, knowing the scale of the map is essential - it is required for specifying a flight path in euclidean space, calculating appropriate control commands and fusing the visual pose estimate with IMU measurements.

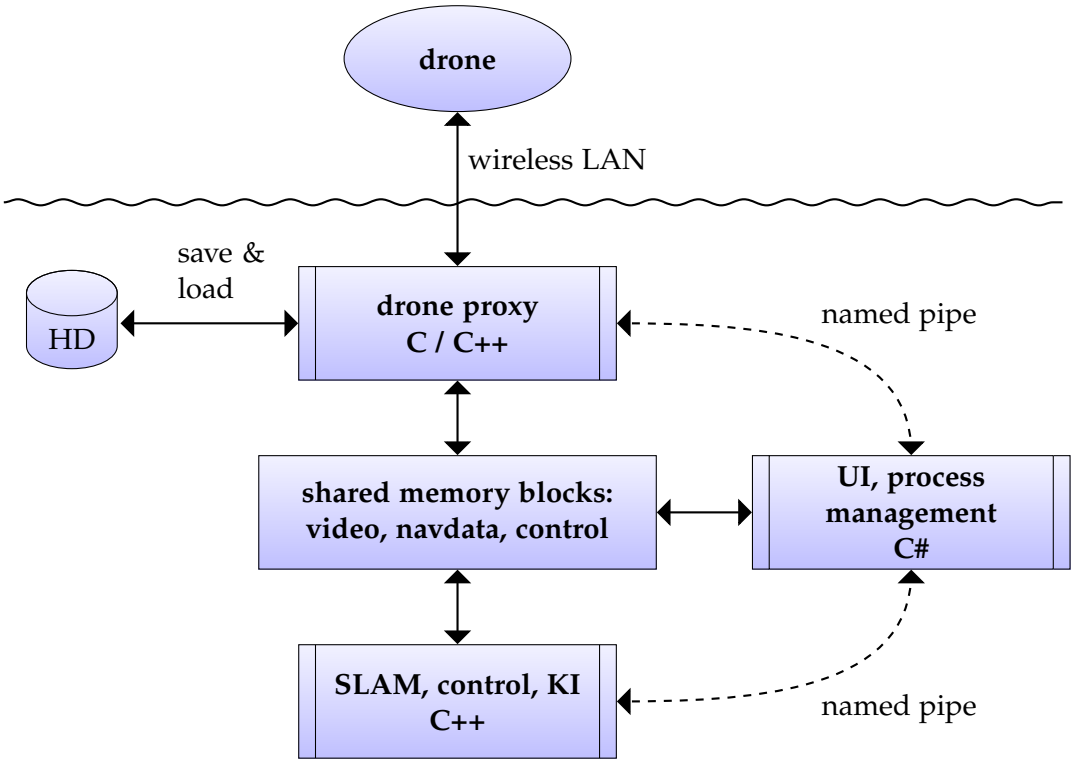


Figure 7.2. The software architecture.

The scale is estimated using the method derived in Chapter 6: a new three-dimensional sample pair is generated every second, using the ultrasound height measurements and integrating over the measured horizontal speeds. If the measured height changes by more than a fixed threshold in between two measurements (corresponding to a vertical speed of more than $3\,\mathrm{m/s}$), a discontinuity in the ground level is assumed and the respective vertical distances are removed. Furthermore the vertical differences are weighted more, as they are more accurate than the measured horizontal velocities. The estimated scaling factor λ^* is applied to PTAM's estimate of the drone's position, i.e. $(x,y,z)^T_{\text{World}} = \lambda^* \, (x,y,z)^T_{\text{PTAM}} + \mathbf{o}$, where the offset $\mathbf{o} \in \mathbb{R}^3$ is adapted for each re-estimation of λ^*, such that for some fix point (e.g. the current drone position, or the starting position) this mapping remains fixed.

7.3.2. Integration of Sensor Data

In addition to scaling and orientating the map correctly, the IMU measurements available are used to increase both resilience and accuracy of PTAM. The Kalman filter is used to generate an accurate prior estimate of the drone's position for each video frame, which is then utilized in three ways:

- it is used as initialization for tracking ($\mathcal{C}_0$), replacing PTAM's built-in decaying velocity model.
- when tracking is lost, it is used as alternative initialization for the recovery process, in some cases speeding up recovery.

- when the roll or pitch angle measured by the IMU deviate strongly from PTAM's pose estimate, tracking is assumed to be lost and the respective result discarded, in particular it is not added as keyframe. This basic validity check drastically reduces the probability of permanently corrupting the map by adding false keyframes.

7.4. State Estimation and Prediction

Estimating the state of the drone - that is its pose as well as velocity - is essential for navigation and autonomous flying. In particular the estimate is required to not only be accurate, but also afflicted with as little delay as possible: the more recent the estimate, the quicker the PID-controller can react and the more accurately the drone can be controlled. We use an **extended Kalman filter** to estimate the state of the drone, fusing optical pose estimates provided by PTAM with sensor measurements provided by the IMU. In this section, we describe the Kalman filter used, in particular we define the state space as well as the state transition model, the observation model and the control model. We also describe how the different sensors are synchronized and how the model parameters are determined.

7.4.1. The State Space

The internal state of the Kalman filter is defined to be

$$\mathbf{x}(t) := (x, y, z, \dot{x}, \dot{y}, \dot{z}, \Phi, \Theta, \Psi, \dot{\Psi})^T \in \mathbb{R}^{10} \tag{7.1}$$

where

- x, y and z correspond to the world-coordinates of the drone center in meters,
- $\dot{x}, \dot{y}$ and $\dot{z}$ correspond to the velocity of the drone in meters per second, expressed in the world coordinate system,
- Φ, Θ and Ψ correspond to roll angle, pitch angle and yaw angle in degree, representing the drone's orientation. While in general such a representation of a 3D-orientation is problematic due to ambiguities and loss of one degree of freedom at certain constellations (Gimbal lock), the fact that both roll and pitch angle are always small makes this representation well suited.
- $\dot{\Psi}$ corresponds to the yaw-rotational speed in degree per second.

As the state changes over time and measurements are integrated in irregular intervals, the respective values will be treated as continuous functions of time when appropriate. For better readability, the time argument is omitted when clear from context.

7.4.2. The Observation Model

The observation model calculates the expected measurements based on the current state of the drone. As two distinct and asynchronous observation sources are available, two separate observation models are required. We adopt the notation introduced in Chapter 4, that

is $\mathbf{z} \in \mathbb{R}^m$ denotes the measurement vector, while $h\colon \mathbb{R}^n \to \mathbb{R}^m$ denotes the observation model, mapping the current state to the respective expected observation.

Visual SLAM Observation Model

The visual SLAM system generates an estimate of the drone's pose for every video frame, that is approximately every 55 ms, which due to the very limited resolution of 320×240 pixel is subject to significant noise. This pose estimate is treated as direct observation of the respective state parameters, that is

$$h_{\text{PTAM}}(\mathbf{x}) := (x, y, z, \Phi, \Theta, \Psi)^T \in \mathbb{R}^6 \tag{7.2}$$

and

$$\mathbf{z}_{\text{PTAM}} := \log(\mathbf{E}_{\mathcal{DC}}\mathbf{E}_{\mathcal{C}}) \in \mathbb{R}^6 \tag{7.3}$$

where $\mathbf{E}_{\mathcal{C}} \in \text{SE(3)}$ is the camera pose as estimated by PTAM, $\mathbf{E}_{\mathcal{DC}} \in \text{SE(3)}$ the (constant) transformation transforming the camera coordinate system to the drone coordinate system, and $\log : \text{SE(3)} \to \mathbb{R}^6$ the transformation from an element of SE(3) to the respective roll-pitch-yaw representation $(x, y, z, \Phi, \Theta, \Psi)$ as described in appendix A.

Sensor Observation Model

As detailed in Chapter 2, the drone sends updated IMU measurements every 5 ms. These measurements however do not correspond to raw sensor measurements, but have already been subject to filtering and other preprocessing steps - as a result, they heavily violate the assumption of independent measurement noise. To compensate for this, the respective measurement variances are chosen to be comparatively high. The sensor values used are:

- **horizontal speed** $\dot{x}_d, \dot{y}_d$, measuring the forward and sideway movement of the drone in its own coordinate frame,
- **relative height** h: this value corresponds to the drone's relative height, and is measured every 40 ms. Assuming a flat ground surface with occasional discontinuities, changes corresponding to a vertical speed of more than $3\,\text{m/s}$ are filtered out. Nevertheless, treating it as direct observation of the drone's height would be problematic: Deviations due to an uneven ground or an inaccurately estimated scale λ would cause unstable and oscillating values for the drone's estimated height and in particular its vertical speed. Instead, only the change in height is used: Let $z(t - \delta t)$ be the height of the drone according to the filter after the last height-observation, and $h(t - \delta t)$ and $h(t)$ the measured relative heights at time $t - \delta t$ and t. The observed absolute height at time t is then given by $z(t - \delta_t) + h(t) - h(t - \delta_t)$.
- **roll- and pitch angles** $\hat{\Phi}, \hat{\Theta}$: as these sensor values are very accurate, drift-free and calibrated with with respect to PTAM's map after the initialization procedure, they can be treated as direct observations of the respective state variables.
- **Yaw angle** $\hat{\Psi}$: as this sensor value is subject to significant drift over time, it handled the same way as the relative height measurements.

The resulting measurement vector and observation function are:

$$h_{\text{IMU}}(\mathbf{x}) := \begin{pmatrix} \cos(\Psi)\dot{x} - \sin(\Psi)\dot{y} \\ \sin(\Psi)\dot{x} + \cos(\Psi)\dot{y} \\ z \\ \Phi \\ \Theta \\ \Psi \end{pmatrix} \in \mathbb{R}^6 \tag{7.4}$$

and

$$\mathbf{z}_{\text{IMU}} := \begin{pmatrix} \dot{x}_d \\ \dot{y}_d \\ z(t-\delta_t) + h(t) - h(t-\delta_t) \\ \hat{\Phi} \\ \hat{\Theta} \\ \Psi(t-\delta_t) + \hat{\Psi}(t) - \hat{\Psi}(t-\delta_t) \end{pmatrix} \in \mathbb{R}^6 \tag{7.5}$$

7.4.3. The State Transition Model

The state transition model $f\colon \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^n$ propagates the state through time, that is based on the state $\mathbf{x}(t)$ and the active control command $\mathbf{u}(t)$ at time t, a prediction for the state $\mathbf{x}(t+\delta_t)$ at time $t+\delta_t$ is calculated. As the drone sends accurate timestamps in microseconds with each sensor measurement package, these can be used to determine the exact prediction intervals δ_t. Prediction is thus performed in irregular time steps of $\delta_t \approx 5 \pm 1\,\text{ms}$. In this section, we describe the state transition model and the control model used, and how it is derived. In Section 7.4.5 we describe how the required model constants are estimated from test-flights.

Horizontal Acceleration

The horizontal velocity of the drone changes according to the horizontal acceleration, which depends on the current attitude of the drone. It is given by

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \end{pmatrix} = \frac{1}{m}\mathbf{f} \tag{7.6}$$

where $m \in \mathbb{R}$ is the drone's mass and $\mathbf{f} \in \mathbb{R}^2$ the sum of all horizontal forces acting upon it. The horizontally acting force $\mathbf{f}$ consists of two components:

- **air resistance $\mathbf{f}_{\text{drag}} \in \mathbb{R}^2$**: for a comparatively slow-moving object, the force acting upon it due to air resistance is approximately proportional to its current velocity, that is
$$\mathbf{f}_{\text{drag}} \propto -\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} \tag{7.7}$$
- **accelerating force due to roll- and pitch angle $\mathbf{f}_{\text{thrust}} \in \mathbb{R}^2$**: the propellers are assumed to generate a constant force acting along the drone's z-axis. If it is tilted, a

portion of this force acts horizontally, this portion is given by projecting the drone's z-axis onto the horizontal plane:

$$\mathbf{f}_{\text{thrust}} \propto \begin{pmatrix} \cos\Psi \sin\Phi \cos\Theta - \sin\Psi \sin\Theta \\ -\sin\Psi \sin\Phi \cos\Theta - \cos\Psi \sin\Theta \end{pmatrix} \tag{7.8}$$

Assuming $\mathbf{f}_{\text{drag}}$ and $\mathbf{f}_{\text{thrust}}$ to be constant over the short time period considered, substituting (7.7) and (7.8) into (7.6) and merging the proportionality constants, the drone's horizontal acceleration is given by

$$\begin{aligned} \ddot{x}(\mathbf{x}) &= c_1 \left(c_2 \left(\cos\Psi \sin\Phi \cos\Theta - \sin\Psi \sin\Theta\right) - \dot{x}\right) \\ \ddot{y}(\mathbf{x}) &= c_1 \left(c_2 \left(-\sin\Psi \sin\Phi \cos\Theta - \cos\Psi \sin\Theta\right) - \dot{y}\right) \end{aligned} \tag{7.9}$$

where c_1 and c_2 are model constants: c_2 defines the maximal speed attained with respect to a given attitude, while c_1 defines how fast the speed adjusts to a changed attitude. The drone is assumed to behave the same in x and y direction.

Influence of Control Commands

The control command $\mathbf{u} = (\bar{\Phi}, \bar{\Theta}, \bar{\dot{\Psi}}, \bar{\dot{z}})^T \in [-1, 1]^4$ defines the desired roll and pitch angles, the desired yaw rotational speed as well as the desired vertical speed of the drone as a fraction of the maximal value permitted. These parameters serve as input values for a controller running onboard on the drone, which then adjusts the engine speeds accordingly. The behavior of this controller is modeled by approximating the roll and pitch rotational speed, the vertical acceleration as well as the yaw rotational acceleration based on the current state and the sent control command. We use a model similar to the one derived for the horizontal accelerations, that is

$$\begin{aligned} \dot{\Phi}(\mathbf{x}, \mathbf{u}) &= c_3 \left(c_4 \bar{\Phi} - \Phi\right) \\ \dot{\Theta}(\mathbf{x}, \mathbf{u}) &= c_3 \left(c_4 \bar{\Theta} - \Theta\right) \\ \ddot{\Psi}(\mathbf{x}, \mathbf{u}) &= c_5 \left(c_6 \bar{\dot{\Psi}} - \dot{\Psi}\right) \\ \ddot{z}(\mathbf{x}, \mathbf{u}) &= c_7 \left(c_8 \bar{\dot{z}} - \dot{z}\right) \end{aligned} \tag{7.10}$$

where c_3 to c_8 are model constants which are determined experimentally in Section 7.4.5. Again, the behavior of the drone is assumed to be the same with respect to roll and pitch angle.

The Complete State Transition Function

The complete state update $\mathbf{x}(t+\delta_t) \leftarrow f(\mathbf{x}(t), \mathbf{u}(t))$ for a time period of δ_t is given by:

$$\begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \Phi \\ \Theta \\ \Psi \\ \dot{\Psi} \end{pmatrix} \leftarrow \begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \Phi \\ \Theta \\ \Psi \\ \dot{\Psi} \end{pmatrix} + \delta_t \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x}(\mathbf{x}) \\ \ddot{y}(\mathbf{x}) \\ \ddot{z}(\mathbf{x}, \mathbf{u}) \\ \dot{\Phi}(\mathbf{x}, \mathbf{u}) \\ \dot{\Theta}(\mathbf{x}, \mathbf{u}) \\ \dot{\Psi} \\ \ddot{\Psi}(\mathbf{x}, \mathbf{u}) \end{pmatrix} \tag{7.11}$$

where $\ddot{x}(\mathbf{x})$ and $\ddot{y}(\mathbf{x})$ are defined in (7.9) and $\dot{\Phi}(\mathbf{x}, \mathbf{u})$, $\dot{\Theta}(\mathbf{x}, \mathbf{u})$, $\ddot{\Psi}(\mathbf{x}, \mathbf{u})$ and $\ddot{z}(\mathbf{x}, \mathbf{u})$ in (7.10).

7.4.4. Time Synchronization

An important aspect when fusing data from multiple sensors - in particular when transmitted via wireless LAN and therefore subject to significant time delay - is the synchronization of this data. In case of the Parrot AR.Drone, in particular the video stream and the resulting pose-estimates from the visual SLAM system are subject to significant time delay: The time required between the instant a frame is captured and the instant the respective calculated control signal is applied (i.e. the time required for encoding the image on the drone, transmitting it via wireless LAN, decoding it on the PC, applying visual tracking, data fusion and control calculations and transmitting the resulting control signal back to the drone) lies between 150 ms and 400 ms. In this section, we describe the communication model and how the resulting delays are compensated for.

Communication Model

When a connection to the drone is established, two latency values are measured using an echo signal to the drone: the time required to transmit an average video frame $t_{10\text{kB}}$, and the time required to transmit a control command or navigational data package $t_{0.5\text{kB}}$. They are updated in regular intervals while the connection is active. $t_{0.5\text{kB}}$ typically lies between 5 ms and 80 ms, while $t_{10\text{kB}}$ lies between 30 ms and 200 ms, depending on the bandwidth used by nearby wireless LAN networks. In order to analyze the communication process, we define the following time spans:

- **camera image delay** $t_{\mathbf{cam}}$: the delay between the instant a frame is captured and the instant it is decoded and ready for further processing on the PC.
- **height and horizontal velocity delay** $t_{\mathbf{xyz}}$: the delay between the instant the data from which the drone estimates its own velocity was measured, and the instant the estimates are available on the PC. The height-measurement has approximately the same delay as the horizontal velocities.

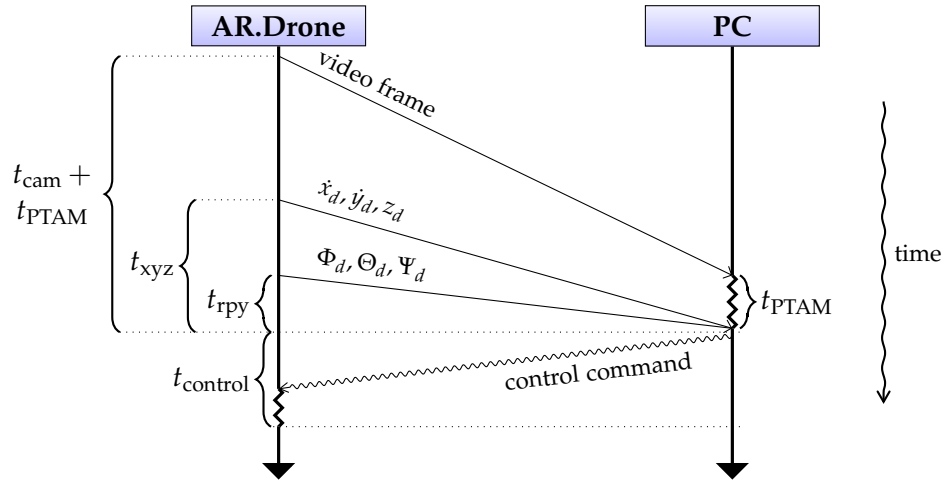


Figure 7.3. Sequence diagram of the communication between the drone and a PC, the defined delays are indicated. For every communication channel, only one exemplary message is drawn - in practice a new video frame is sent approximately every 55 ms, new sensor measurements every 5 ms and an updated control command every 10 ms.

- **gyroscope data delay t_{rpy}**: the measurements of the drone's roll, pitch and yaw angle are subject to little preprocessing and therefore have the smallest delay, which is denoted by t_{rpy}.
- **control delay t_{control}**: every 10 ms an updated control signal is calculated by the PC, sent to the drone and used as setpoint for the internal attitude controller. The time between the instant it is sent, and the instant it is applied is denoted by t_{control}.
- **tracking time t_{PTAM}**: the time required for visual tracking, which varies significantly from frame to frame.

Figure 7.3 depicts the communication between the drone and a PC as a sequence diagram, the defined delays are indicated.

Compensating for Delays

All sensor values received from the drone are stored in a buffer, as are all control signals sent. They are not incorporated into the Kalman filter directly. Only when a new video frame is received, the Kalman filter is rolled forward permanently up to the point in time at which it was received (T_1), incorporating time-corrected sensor measurements as observations and removing them from the buffer. The resulting pose is used as initialization for visual tracking. After the tracking is completed, the calculated pose is added as observation to the Kalman filter.

Simultaneously - every 10 ms - a new control command is calculated using the best estimate possible of the drone's state for the point in time the command will be applied, i.e. t_{control} in the feature (T_3): The Kalman filter is temporarily rolled forward up to T_3, incorporating all sensor measurements and sent control commands stored in the buffers. This prediction process is depicted in Figure 7.4.

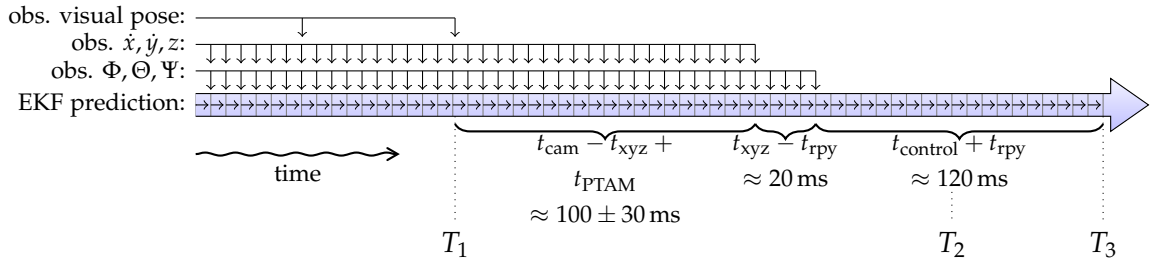


Figure 7.4. The filter is permanently rolled forward up to T_1. For each new control command, it is temporarily rolled forward up to T_3, which is the point in time at which the calculated control command will take effect: For the first $t_{\text{cam}} - t_{\text{xyz}} + t_{\text{PTAM}} \approx 100 \pm 30\,\text{ms}$, complete IMU measurements are available and can be added as observations. For the following $t_{\text{xyz}} - t_{\text{rpy}} \approx 20\,\text{ms}$, only roll, pitch and yaw angle measurements are available. The last $t_{\text{control}} + t_{\text{rpy}} \approx 120\,\text{ms}$ are predicted using only the previously sent control commands. This calculation process is done approximately at T_2. The given time spans are typical values, in practice they are adapted continuously corresponding to the measured wireless LAN latencies.

Measuring the Delays

As the Parrot AR.Drone does not transmit timestamps with the sent video frames or sensor data packages, only the time at which they were received by the PC can be used to experimentally determine the values required in order to compensate for the delays present.

- $(t_{\text{cam}} - t_{\text{rpy}})$ is estimated by comparing the roll angle measured by the IMU with the roll angle calculated by PTAM over a series of test flights: Let $\Phi_{\text{PTAM}}(t)$ be the roll angle measured by PTAM for a frame received at time t, and $\Phi_{\text{IMU}}(t)$ the roll angle measured by the IMU, received at time t. The delay is estimated by minimizing the sum of squared differences between the two functions (interpolating linearly), sampling at regular intervals of 5 ms:

$$\begin{aligned} t_{\text{cam}} - t_{\text{rpy}} &= \arg\min_{\delta_t} E(\delta_t) \\ \text{with} \quad E(\delta_t) &:= \sum_t (\Phi_{\text{IMU}}(t) - \Phi_{\text{PTAM}}(t + \delta_t))^2 \end{aligned} \tag{7.12}$$

  The result is visualized in Figure 7.5.

- $(t_{\text{cam}} - t_{\text{xyz}})$ is estimated by comparing the height measured by the IMU with the height calculated by PTAM over a series of test flights, using the same method.

- $(t_{\text{rpy}} + t_{\text{control}})$ is estimated by comparing the sent control commands with the roll angle measured by the IMU, and manually aligning the instant the control command is changed with the instant this change starts to take effect. Figure 7.6 shows a short extract of the test flight.

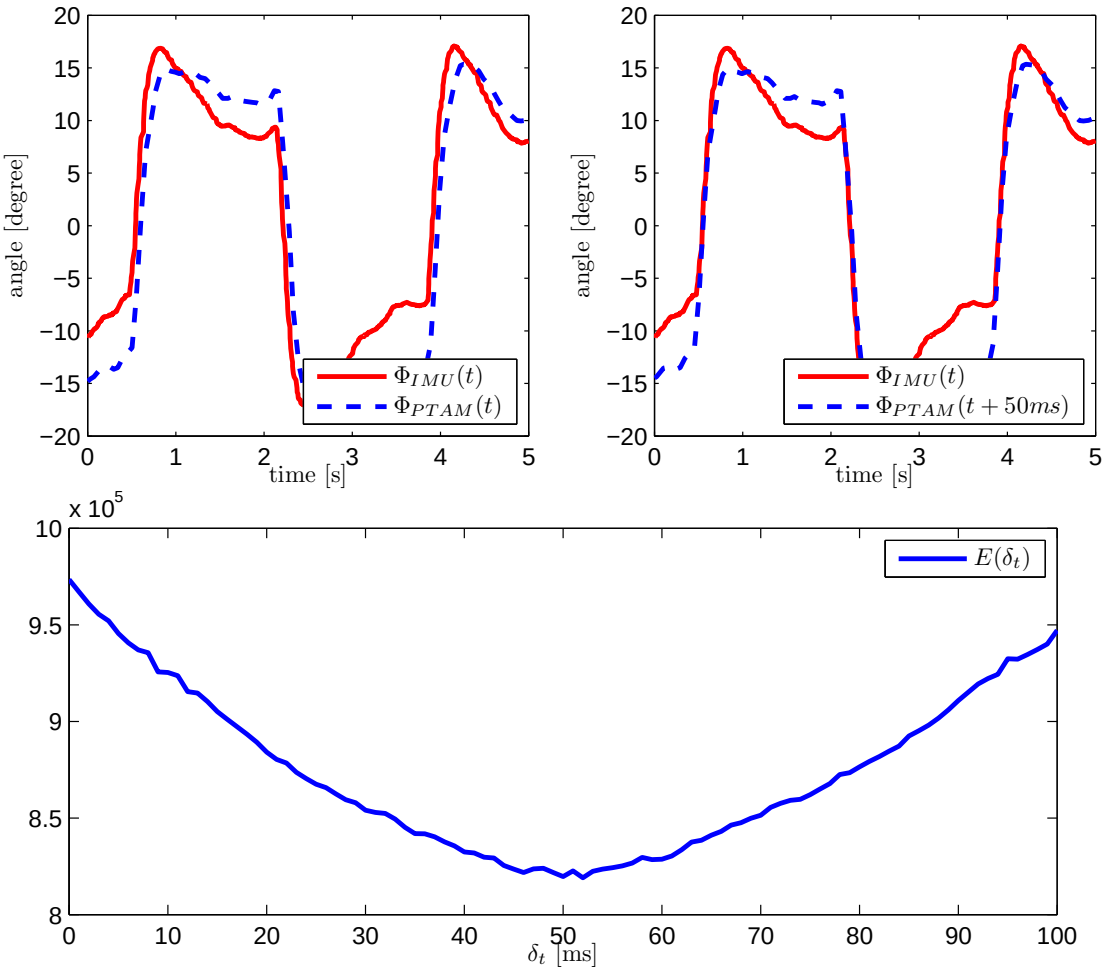


Figure 7.5. The top-left plot shows the measured angles $\Phi_{\text{PTAM}}(t)$ and $\Phi_{\text{IMU}}(t)$ for an extract of the test flights, the delay is clearly visible. In the right plot, the same extract is plotted time-aligned. The bottom plot shows the error function $E(\delta_t)$, a clear minimum at $\delta_t = 50\,\text{ms}$ becomes visible.

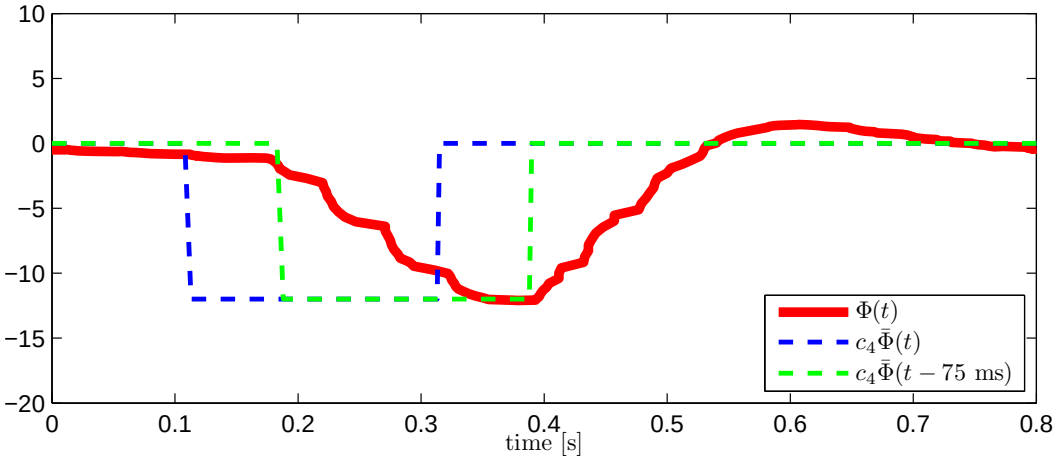


Figure 7.6. Manual alignment of the sent control signal and the respective attitude angle.

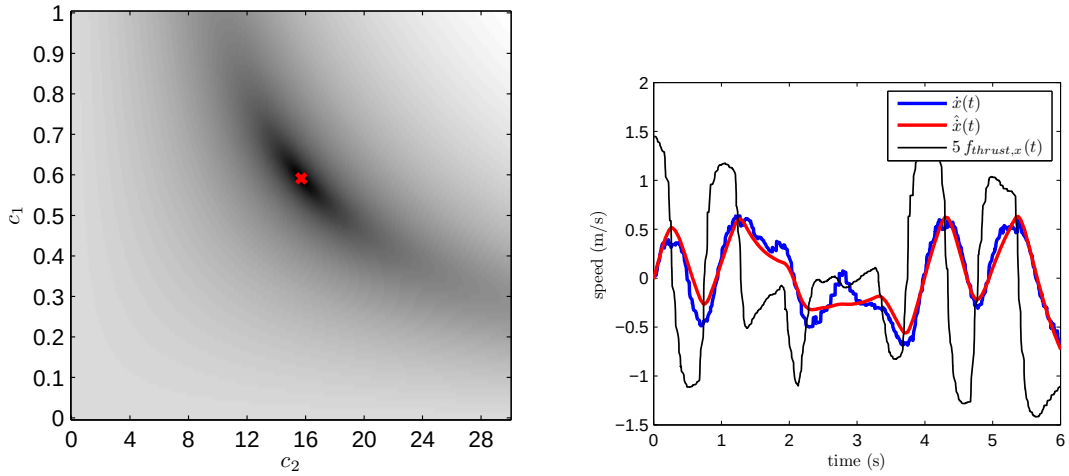


Figure 7.7. Left: visualization of $E_{c_1,c_2}(c_1, c_2)$ as defined in (7.14): a clear minimum at $c_1 = 0.6$ and $c_2 = 16$ is visible. Right: true speed $\dot{x}(t)$ versus predicted speed $\hat{\dot{x}}(t)$, as well as $\mathbf{f}_{\text{thrust},x}(t)$ for a short extract of the test-flights.

Taking into account the known latencies for the test flights $t_{10\text{kB}} = 35\,\text{ms}$ and $t_{0.5\text{kB}} = 5\,\text{ms}$, we determined the following delay values. Note that neither of the four defined time spans t_{cam}, t_{rpy}, t_{xyz} and t_{control} can be determined from these three relations alone, however only the following values are required:

$$\begin{aligned} (t_{\text{cam}} - t_{\text{rpy}}) &= 20\,\text{ms} + t_{10\text{kB}} - t_{0.5\text{kB}} \\ (t_{\text{cam}} - t_{\text{xyz}}) &= 0\,\text{ms} + t_{10\text{kB}} - t_{0.5\text{kB}} \\ (t_{\text{rpy}} + t_{\text{control}}) &= 65\,\text{ms} + 2\,t_{0.5\text{kB}} \end{aligned} \tag{7.13}$$

7.4.5. Calibration of Model Parameters

The constant parameters of the model derived above, c_1 to c_8, were estimated by minimizing the difference between the values predicted by the model and the true values calculated by the EKF over a series of test flights. This is possible because the influence of c_1 to c_8 on the state are negligible after the respective observations have been integrated (at T_1 in Figure 7.4). They are only required for the state prediction at T_3. For c_1 and c_2 we minimize the following error function:

$$\begin{aligned} E_{c_1,c_2}(c_1, c_2) &:= \sum_t \left(\dot{x}(t) - \hat{\dot{x}}(t)\right)^2 \\ \hat{\dot{x}}(t + \delta_t) &= \hat{\dot{x}}(t) + \delta_t\, c_1\, (c_2\, \mathbf{f}_{\text{thrust},x}(t) - \hat{\dot{x}}(t)) \end{aligned} \tag{7.14}$$

Observe that $\hat{\dot{x}}(t)$ is calculated using solely the measured attitude of the drone, while $\dot{x}(t)$ is the "ground truth", that is the speed estimate at time t after integrating all measurements available. The result is visualized in Figure 7.7. The remaining model parameters c_3 to c_8 were estimated analogously, see Figures 7.8 and 7.9 for the results.

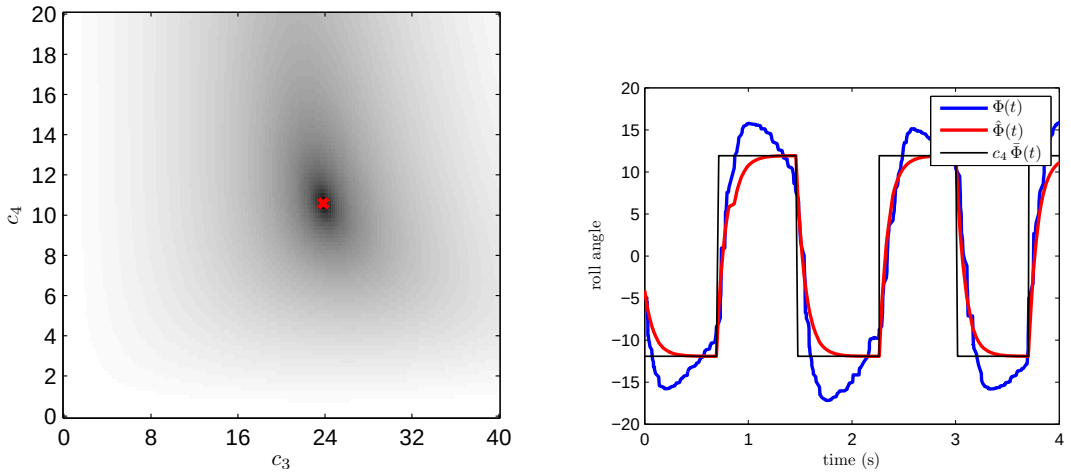


Figure 7.8. Left: visualization of $E_{c_3,c_4}(c_3, c_4)$: a clear minimum at $c_3 = 10.6$ and $c_4 = 24$ is visible. Right: true roll angle $\Phi(t)$ versus predicted roll angle $\hat{\Phi}(t)$, as well as the respective control command $\bar{\Phi}(t)$ for a short extract of the test-flights.

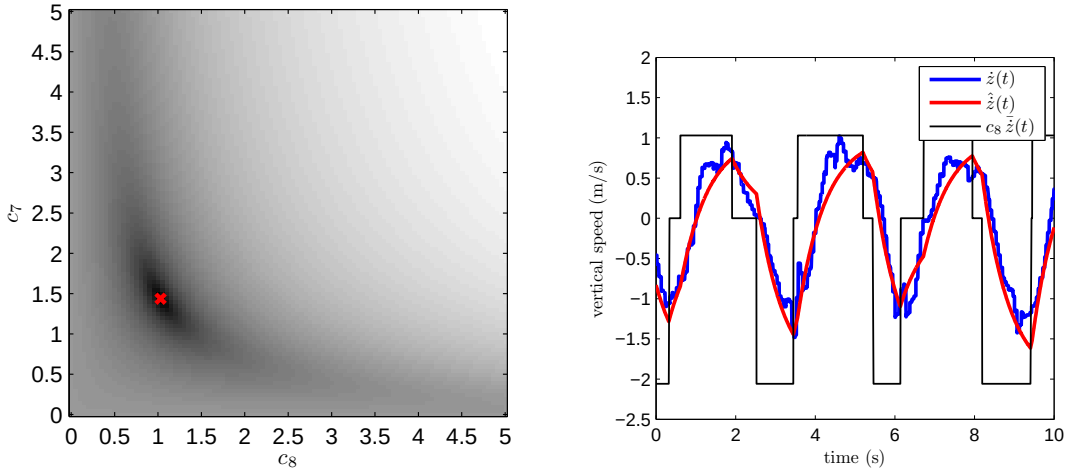


Figure 7.9. Left: visualization of $E_{c_7,c_8}(c_7, c_8)$: a clear minimum at $c_7 = 1.4$ and $c_8 = 1$ is visible. Right: true vertical speed $\dot{z}(t)$ versus predicted vertical speed $\hat{\dot{z}}(t)$, as well as the respective control command $\bar{\dot{z}}(t)$ for a short extract of the test-flights.

7.5. Drone Control

In order to control the drone we use a PID controller, taking the predicted drone state for T_3 as input. In particular we directly use the speed estimates maintained by the Kalman filter. Let $\mathbf{x} = (x, y, z, \dot{x}, \dot{y}, \dot{z}, \Phi, \Theta, \Psi, \dot{\Psi})^T$ be the predicted state of the drone, and $\mathbf{p} = (\hat{x}, \hat{y}, \hat{z}, \hat{\Psi})^T$ the target position and yaw angle. The control signal $\mathbf{u} = (\bar{\Phi}, \bar{\Theta}, \bar{\dot{z}}, \bar{\dot{\Psi}})$ is now calculated by applying PID control to these four parameters, and rotating the result horizontally such that it corresponds to the drone's coordinate system:

$$\begin{aligned}
\bar{\Phi} &= (0.5\,(\hat{x} - x) + 0.32\,\dot{x})\,\cos\Psi - (0.5\,(\hat{y} - y) + 0.32\,\dot{y})\,\sin\Psi \\
\bar{\Theta} &= -(0.5\,(\hat{x} - x) + 0.32\,\dot{x})\,\sin\Psi - (0.4\,(\hat{y} - y) + 0.32\,\dot{y})\,\cos\Psi \\
\bar{\dot{z}} &= 0.6\,(\hat{z} - z) + 0.1\,\dot{z} + 0.01 \int (\hat{z} - z) \\
\bar{\dot{\Psi}} &= 0.02\,(\hat{\Psi} - \Psi)
\end{aligned} \tag{7.15}$$

We found that integral control is only required for controlling the drone's height, while the yaw angle can be controlled by a proportional controller alone without resulting in oscillations or overshoot. The integral term for the height control is reset when the target height is first reached, and capped at a maximum of 0.2. The other parameters were determined experimentally.

8. Results

In this chapter, we evaluate the performance and accuracy of the developed system on experimental data obtained from a large number of test flights with the AR.Drone. In the first Section 8.1, we verify the accuracy of the scale estimation method derived in Chapter 6. In Section 8.2, we analyze the precision of the prediction model and the time synchronization method as described in Section 7.4, giving both an example as well as a quantitative evaluation. In Section 8.3, we measure the performance of the PID controller as described in Section 7.5 with respect to stability and convergence speed. Section 8.4 demonstrates how the addition of a visual SLAM system significantly improves the control accuracy and eliminates drift. In Section 8.5, the system's ability to recover from a temporary loss of visual tracking due to e.g. fast motion or external influences.

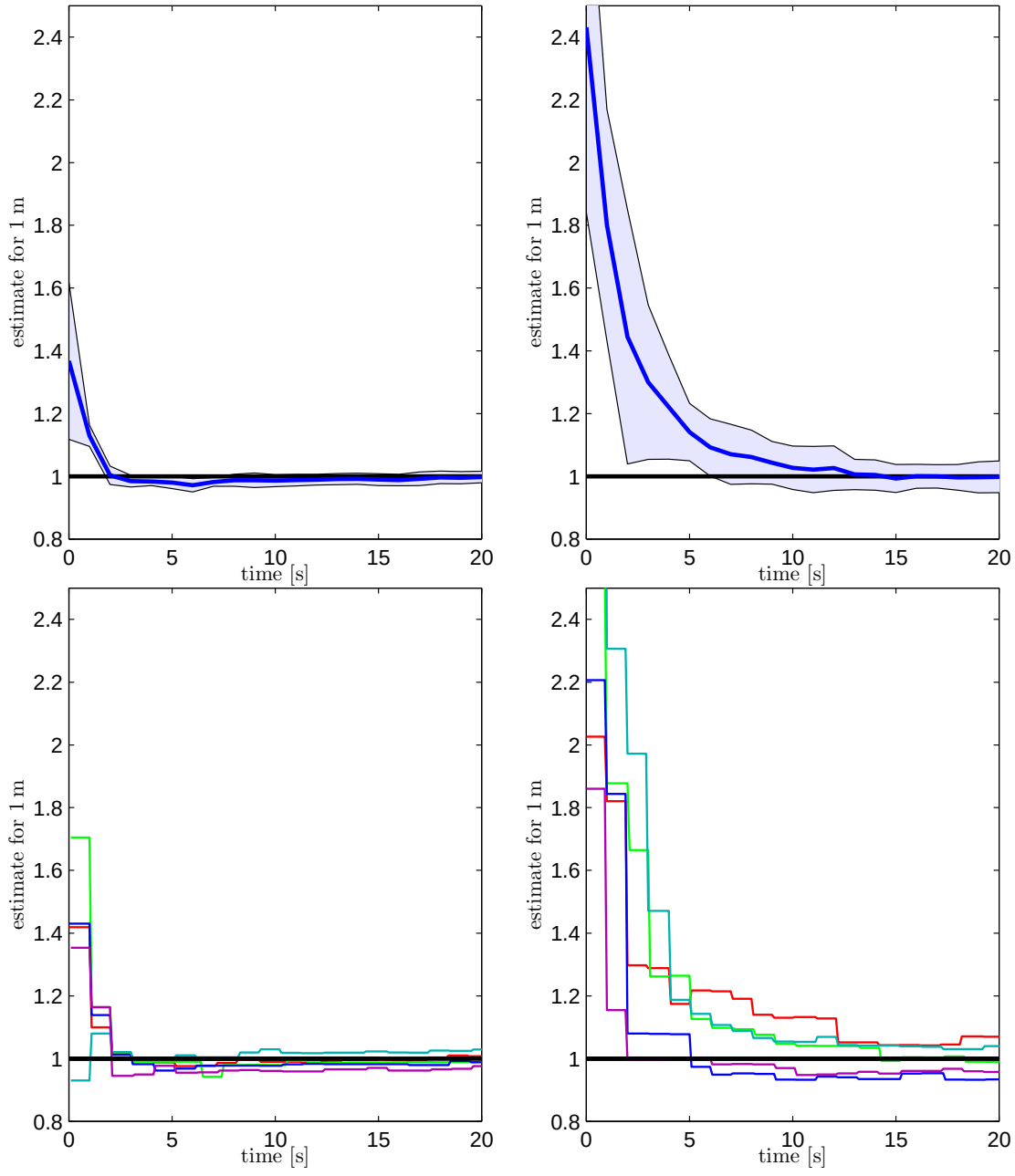
8.1. Scale Estimation Accuracy

To analyze the accuracy of the scale estimation method used and derived, the drone is repeatedly instructed to fly a fixed figure, while the scale of the map is re-estimated every second. The ground truth is obtained afterwards by manually moving the drone a fixed distance, and comparing the moved distance with the displacement measured by the visual SLAM system. Figure 8.1 shows the estimated length of 1 m for the first 20 s of the test flights. For Figure 8.1a, we only use vertical motion to estimate the scale, and instruct the drone to repeatedly fly up and down a distance of 1.5 m. For Figure 8.1b, only horizontal motion is used, and the drone is instructed to repeatedly fly along a horizontal line with a length of 1.5 m.

Observe how the scale can be estimated accurately from both types of motion alone, the estimate however converges faster and is more accurate if the drone moves vertically, i.e. the ultrasound altimeter can be used. For vertical motion, the standard deviation of the estimation error after 20 s is only 1.75%, while for horizontal motion it is 5%. In practice, all three dimensions are used, allowing for accurate scale estimation from arbitrary motion - the accuracy depending on how the drone moves. The initial scale estimate is base only on the distance moved in between the first two keyframes, and therefore is very inaccurate. In oder to avoid severe overshoot due to a wrong scale, the initial estimate is doubled - hence the initially large estimates.

8.2. Prediction Model Accuracy

The main purpose of the prediction and control model $f\colon \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^n$ of the EKF - as detailed in Section 7.4.3 - is to compensate for the time delays present in the system, in particular to allow prediction for the timespan where no sensor measurements at all, or



(a): vertical motion (b): horizontal motion

Figure 8.1. Error on the estimated scale for a total of ten test flights, five with only horizontal and five with only vertical motion. The top plots show mean and standard deviation for the estimated length of 1 m. The bottom plots show the individual runs.

only IMU measurements are available (T_1 to T_3 in Figure 7.4, in practice around 150 ms to 400 ms). Due to the relative inaccuracy of the prediction model compared to the sensor measurements and the visual pose estimates, its influence on the filter state at T_1, after all sensor measurements up to that point in time have been incorporated is negligible. Only the for T_3 predicted pose - which is the relevant one for the PID controller - is strongly influenced by the prediction model.

Qualitative Example

An example of the accuracy of the prediction model with respect to horizontal speed, position and roll angle is given in Figure 8.2: For this plot, the drone is instructed to fly to and hold $x = 0\,\text{m}$. At $t = 5\,\text{s}$, it is forcefully pushed away from this position, and immediately tries to return to it.

The prediction for the drone's state at time t as computed at T_3, $\mathbf{x}_{T_3}(t)$, is drawn as red line. The state of the filter at time t as computed at T_1, $\mathbf{x}_{T_1}(t)$, is drawn as dashed, black line: this is the best state estimate available at time t without predicting ahead, based on the most recent video frame tracked - note the clearly visible delay. The continuous black line shows the same estimate as the dashed one, but shifted in time such that it aligns with the time the respective measurements and video frames were taken at, that is $\mathbf{x}_{T_1}(t + t_d)$. For this comparison it is treated as ground truth, as the state prediction model has comparatively little influence on it as argued above.

Quantitative Evaluation

We also performed a quantitative evaluation of the accuracy gained due to the prediction model. We do this by treating $\mathbf{x}_{T_1}(t + t_d)$ as ground truth, and computing the root mean squared error (RMSE) between this ground truth and

1. the pose computed by the prediction model $\mathbf{x}_{T_3}(t)$ (red line in Figure 8.2).

$$\text{RMSE}_{\text{pred}} := \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{T_1}(t_i + t_{i,d}) - x_{T_3}(t_i))^2} \tag{8.1}$$

2. the most recent estimate available at that point in time, without predicting ahead $\mathbf{x}_{T_1}(t)$ (dashed black line in Figure 8.2).

$$\text{RMSE}_{\text{raw}} := \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{T_1}(t_i + t_{i,d}) - x_{T_1}(t_i))^2} \tag{8.2}$$

Table 8.1 shows the RMSE for three different flight patterns and all state variables. Also given is the percentage decrease of the RMSE when predicting ahead ($\text{acc} := 1 - \frac{\text{RMSE}_{\text{pred}}}{\text{RMSE}_{\text{raw}}}$). While $\text{RMSE}_{\text{pred}}$ is always smaller than RMSE_{raw}, the difference becomes far more significant for flights where the respective parameter changes frequently. Furthermore it can be observed that the prediction model for vertical motion is relatively poor due to the comparatively unpredictable reaction of the drone to thrust control, while the model for

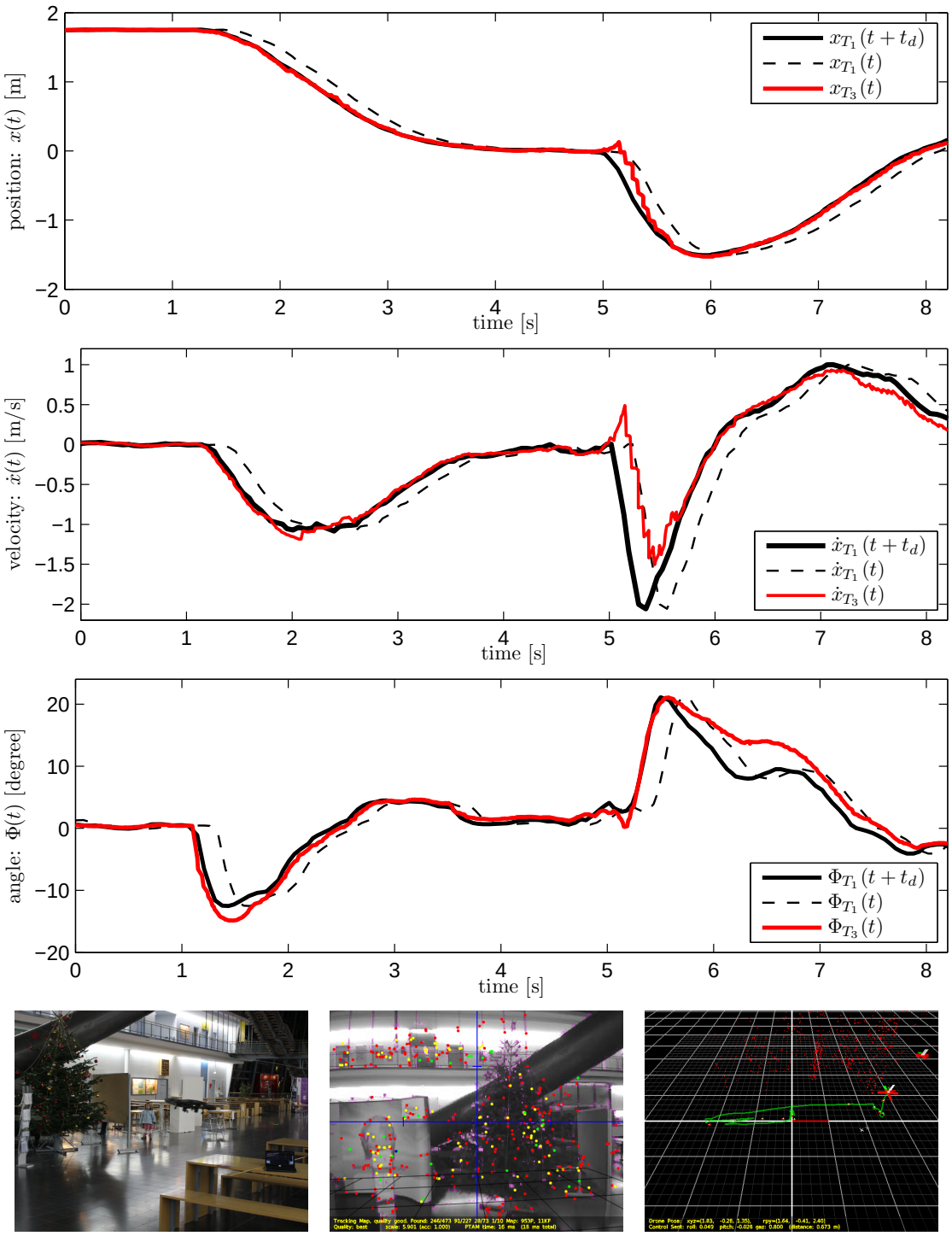


Figure 8.2. The position (top), velocity (middle) and roll angle (bottom) of the drone. The accuracy of the prediction model can well be observed by comparing the red to the black line: Only at $t = 5\,\mathrm{s}$ there is a strong deviation, as the prediction model cannot predict the drone being pushed away - this can only be observed after the respective sensor measurements are added to the filter, hence the red line briefly coincides with the dashed black line. The bottom row shows the environment this experiment was conducted in, the total delay t_d lay around 200 ms.

	Flight pattern 1: horizontal movement			Flight pattern 2: vertical movement			Flight pattern 3: house		
	$RMSE_{raw}$	$RMSE_{pred}$	acc	$RMSE_{raw}$	$RMSE_{pred}$	acc	$RMSE_{raw}$	$RMSE_{pred}$	acc
x	18 cm	2 cm	**87%**	4 cm	1 cm	62%	3 cm	1 cm	64%
y	3 cm	1 cm	59%	3 cm	1 cm	58%	2 cm	1 cm	55%
z	6 cm	4 cm	30%	9 cm	3 cm	**65%**	7 cm	3 cm	63%
$\dot{x}$	27 cm/s	8 cm/s	**70%**	10 cm/s	5 cm/s	49%	8 cm/s	4 cm/s	53%
$\dot{y}$	9 cm/s	4 cm/s	56%	8 cm/s	4 cm/s	51%	4 cm/s	3 cm/s	38%
$\dot{z}$	26 cm/s	23 cm/s	12%	19 cm/s	16 cm/s	**17%**	15 cm/s	13 cm/s	8%
Φ	3.9°	1.7°	57%	1.6°	0.7°	57%	1.3°	0.5°	60%
Θ	1.7°	0.5°	68%	1.4°	0.6°	57%	0.8°	0.3°	58%
Ψ	0.7°	0.2°	67%	0.7°	0.2°	71%	0.3°	0.1°	61%
$\dot{\Psi}$	5.7 °/s	3.2 °/s	45%	5.7 °/s	3.1 °/s	45%	3.1 °/s	1.9 °/s	39%

Table 8.1. Accuracy of the prediction model for three different flight patterns: movement only in x-direction (100 times a distance of 1.5 m, taking 4 minutes in total), movement only in z-direction (100 times a height of 1.5 m, taking 8 minutes in total) and a horizontal house (see Figure 8.6a).

movement in horizontal direction is quite accurate, significantly reducing the RMSE. Note that this evaluation criterion does not capture the full benefit of the prediction model: while for some parameters the prediction is inaccurate or might even increase the error, it compensates very well for delays - which is essential for avoiding oscillations.

8.3. Control Accuracy and Responsiveness

We evaluate the accuracy and responsiveness of the PID-controller with respect to two main criteria:

- **Stability**: How accurately can the drone hold a target position?
- **Convergence speed**: How fast does the drone reach and hold a target position from a certain initial distance?

The performance of the PID controller heavily depends on the accuracy of the estimated drone state - it is particularly sensible to delays, as a delay quickly leads to strong oscillations. Figure 8.3 shows the x, y and z control for the star flown in Figure 8.6c, as well as the respective proportional and derivative components as an example.

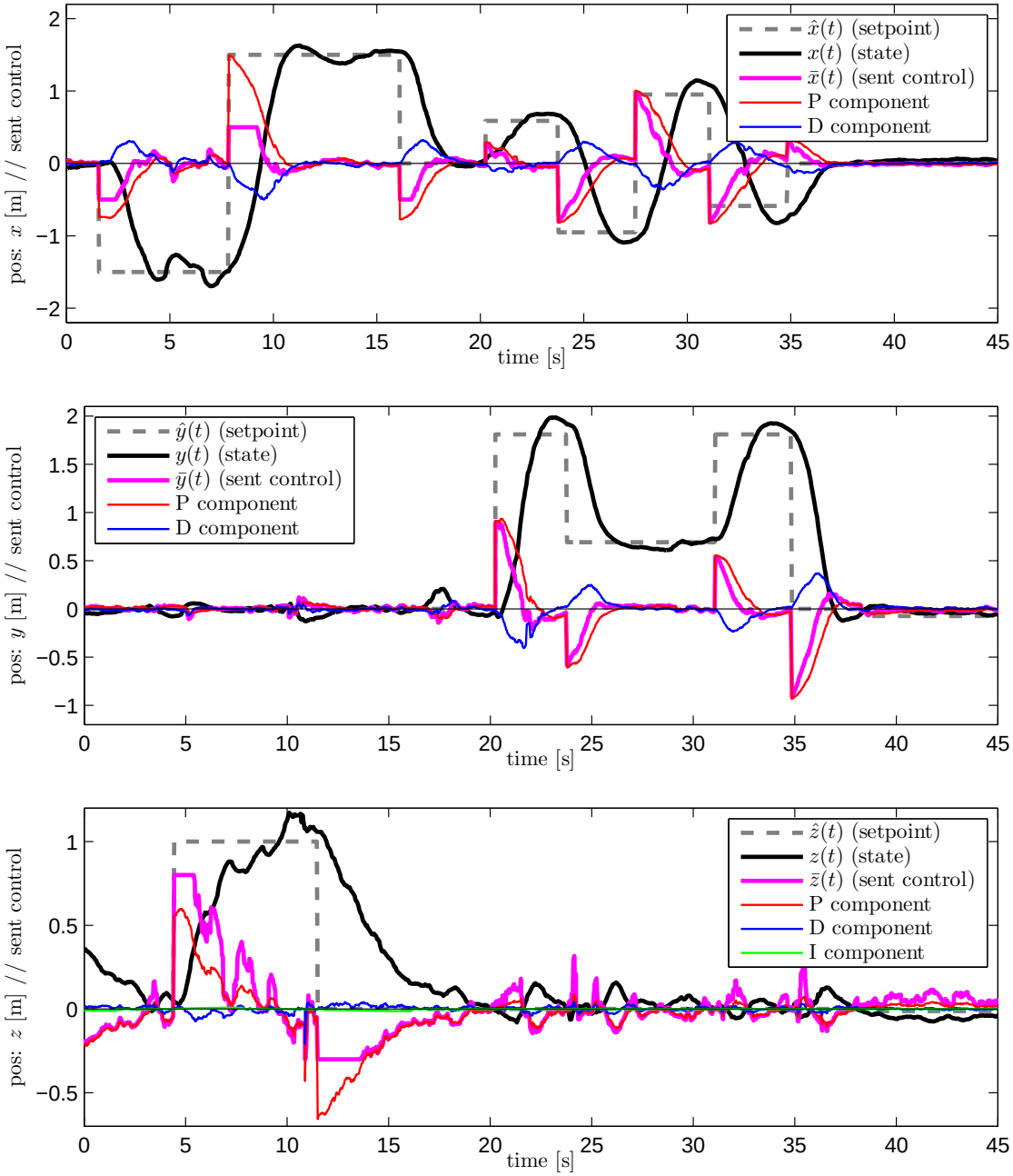


Figure 8.3. Control example: x (top), y (middle) and z (bottom) control for the star flight path displayed in Figure 8.6c, as well as the previously flown vertical calibration rectangle. Initially, the maximum control signal sent is capped at 0.5, as the scale estimate might still be inaccurate, and such that PTAM can take good keyframes without too much motion blur.

	(x, y) RMSE (cm)	(z) RMSE (cm)	(x, y, z) RMSE (cm)	(Ψ) RMSE (degree)
large indoor area	7.7	1.6	7.8	0.30
office	10.3	6.9	12.4	0.65
outdoor	17.6	3.8	18.0	0.72

Table 8.2. Control stability: outdoor, the horizontal RMSE is comparatively large due to wind, while the vertical RMSE in a constrained environment with an uneven floor (office) is poor due to turbulences and unreliable altimeter measurements. The yaw RMSE is very low under all conditions.

Stability

To evaluate the stability of the controller, the drone is instructed to stay at a given target position for 60 s, having initialized the map and its scale by previously flying a 2 m $\times$ 1 m vertical rectangle. As evaluation criterion we use again the root mean square error, sampling in 5 ms intervals. The RMSE is computed according to

$$\text{RMSE}(\mathbf{x}_i, \mathbf{y}_i) := \sqrt{\frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{x}_i} \tag{8.3}$$

assuming the target position to be $\mathbf{0}$. Table 8.2 shows the total, the horizontal and the vertical RMSE in three different environments depicted in Figure 8.4. It can be observed that the drone is capable of very accurately holding its position in different environments, without oscillating.

Convergence Speed

To measure the convergence speed, the drone is instructed to repeatedly fly to and hold a target position at a certain distance $\mathbf{d} \in \mathbb{R}^3$. We examine the following values:

- **reach time** t_{reach}: the time passed until the target is reached for the first time,
- **convergence time** t_{conv}: the time required for achieving a stable state at the target position, that is not leaving it for 5 s,
- **overshoot** d_{over}: the maximal distance from the target, after it has first been reached.

The drone is considered to be at the target position if its Euclidean distance is less than 15 cm. Figure 8.5 shows an example of such a flight, Table 8.3 shows these values for different $\mathbf{d}$.

It can be observed that the drone is capable of moving much faster in horizontal direction (covering 4 m in only 3 s, flying at a speed of up to 2 $^{\text{m}}/_{\text{s}}$) - this is directly due to the low accuracy of the height prediction model (quickly leading to oscillations), and the fact that the drone's reaction to thrust control is difficult to predict.

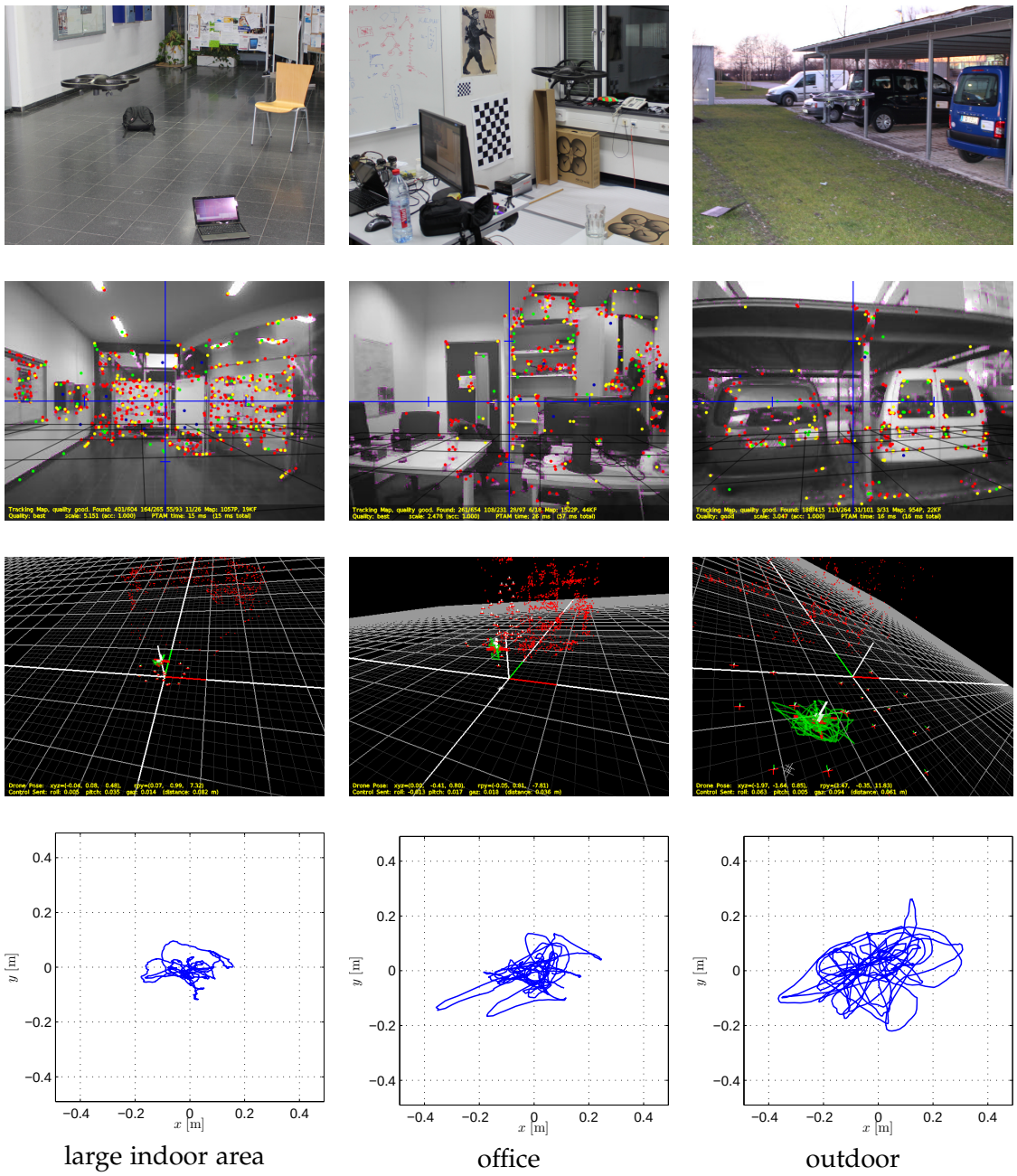


Figure 8.4. The three different environments for the stability evaluation in Table 8.2. The bottom row shows the horizontal path of the drone over 60 s.

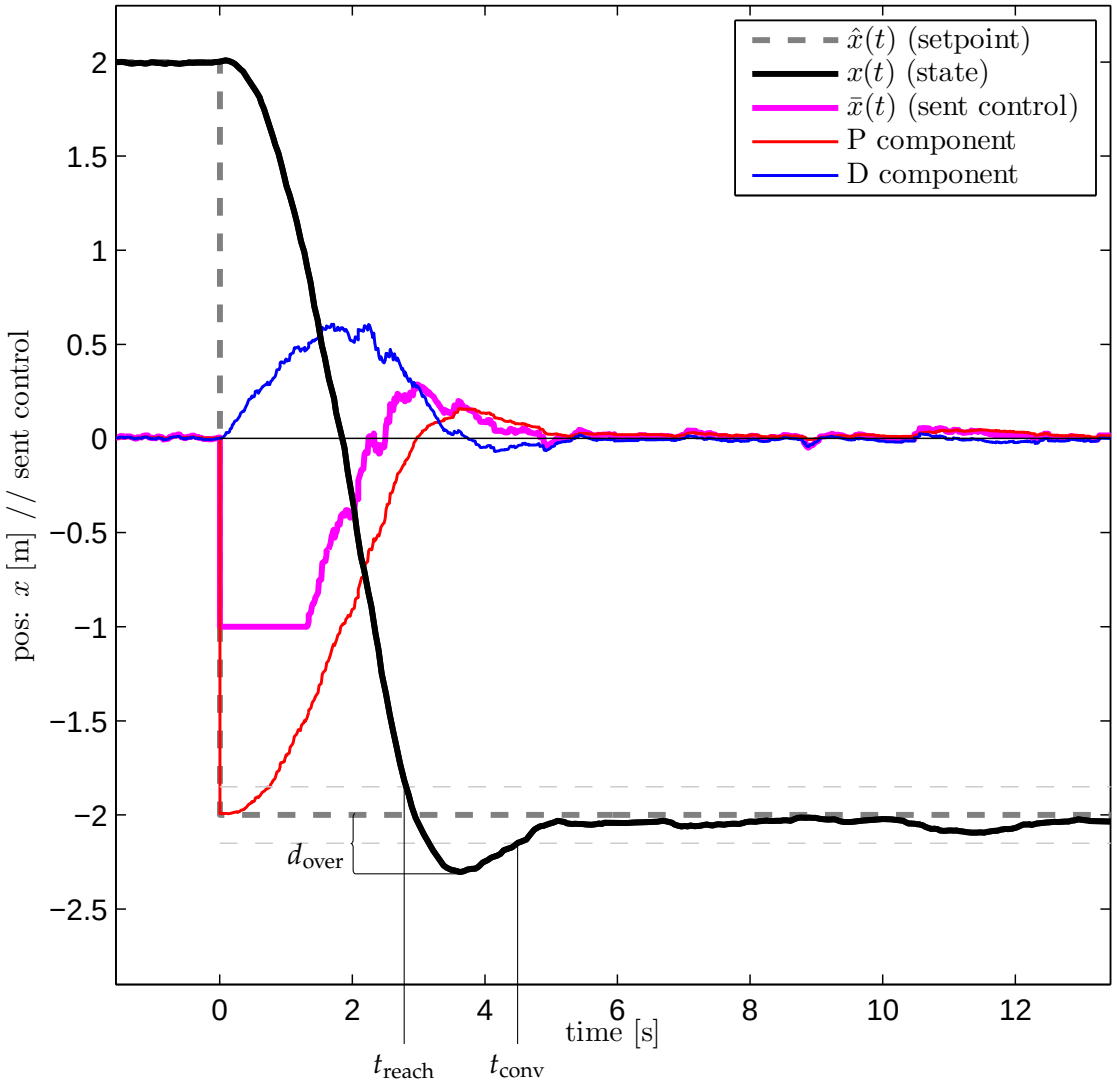


Figure 8.5. Example for flying to and holding a position 4 m away, the four values examined are indicated: $t_{\text{reach}} = 2.7\,\text{s}$, $t_{\text{conv}} = 4.65\,\text{s}$ and $d_{\text{over}} = 30\,\text{cm}$.

	t_{reach} (s)	t_{conv} (s)	d_{over} (cm)
$\mathbf{d} = (1\,\text{m}, 0\,\text{m}, 0\,\text{m})^T$	1.6 ± 0.2	3.1 ± 1.3	10 ± 5
$\mathbf{d} = (4\,\text{m}, 0\,\text{m}, 0\,\text{m})^T$	3.0 ± 0.3	5.5 ± 0.5	26 ± 10
$\mathbf{d} = (0\,\text{m}, 0\,\text{m}, 1\,\text{m})^T$	3.1 ± 0.1	3.1 ± 0.1	2 ± 1
$\mathbf{d} = (1\,\text{m}, 1\,\text{m}, 1\,\text{m})^T$	3.0 ± 0.1	3.9 ± 0.5	10 ± 5

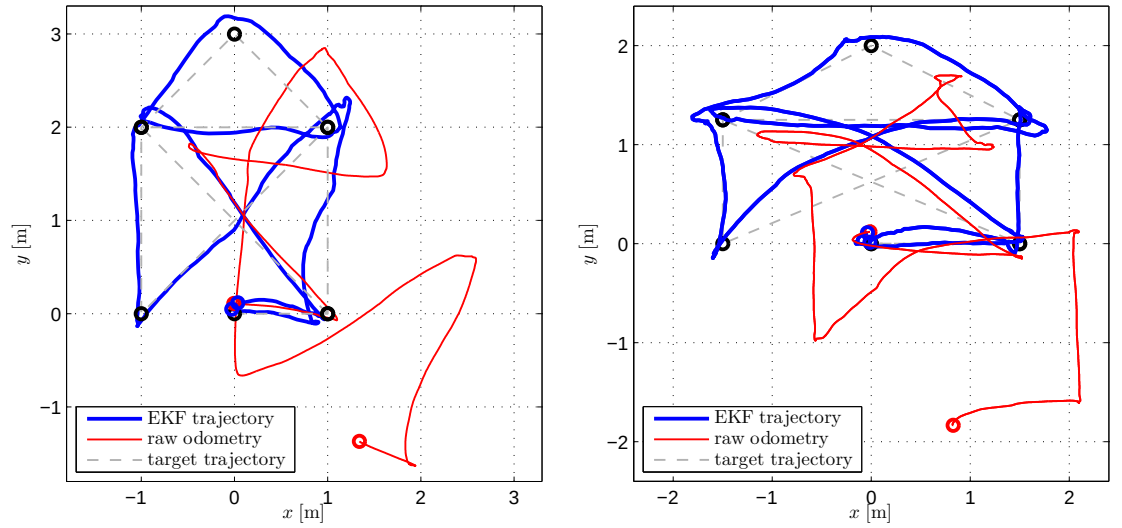
Table 8.3. Control convergence speed: t_{reach}, t_{conv} and d_{over} for different distances. The table shows the average and standard deviations over 10 test flights each, and for 4 different values for $\mathbf{d}$. The drone control behaves very similar in all horizontal directions, we therefore only examine flight along the x direction.

8.4. Drift Elimination due to Visual Tracking

To demonstrate how the incorporation of a visual SLAM system eliminates drift, we compare the estimated trajectory from the EKF (fusing the visual pose estimate with the IMU data) with the raw odometry, that is the trajectory estimated only from the IMU measurements and optical-flow based speed estimates of the drone. Figures 8.6a to 8.6c show three flight paths consisting of way points and flown without human intervention, the map and its scale having been initialized by previously flying a $1\,\mathrm{m} \times 3\,\mathrm{m}$ vertical rectangle. All three flights took approximately 35 s each (15 s for the initialization rectangle and 20 s for the figure itself). In Figure 8.6d, the drone is instructed to hold its position while being pushed away for 50 s. In all four flights the drone landed no more than 15 cm away from its takeoff position.

8.5. Robustness to Visual Tracking Loss

In this section, we demonstrate the system's ability to deal with a temporary loss of visual tracking. This happens not only when the drone is rotated quickly, but also when it is flying at high velocities ($> 1.5\,\mathrm{m/s}$), as the frontal camera is subject to significant motion blur and rolling shutter effects. Figure 8.7 shows the drone being pushed and rotated away from its target position, such that visual tracking inevitably fails. Using only IMU and altimeter measurements, the drone is able to return to a position close enough to its original position, such that visual tracking recovers and it can return to its target position.



(a) house (b) house 2

(c) star (d) under disturbance

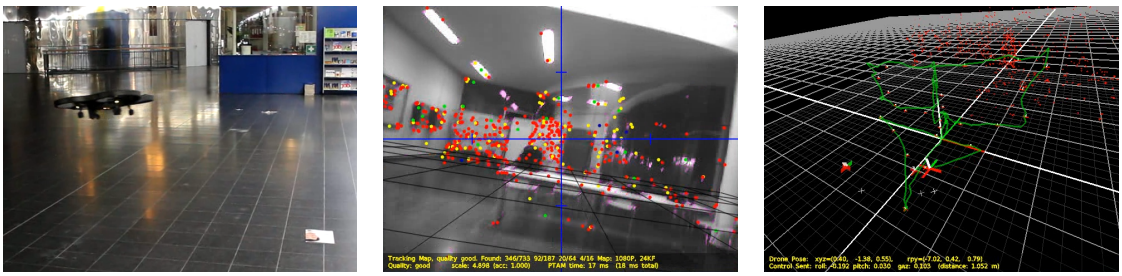


Figure 8.6. (a) to (c): top view of three closed, horizontal figures flown automatically by the drone, the axis are labeled in meters. (d): top view of the drone holding its position, while being pushed away repeatedly. Both the horizontal as well as the yaw-rotational drift of the raw odometry are clearly visible, in particular when the drone is being pushed away. The bottom row shows the environment these experiments were conducted in.

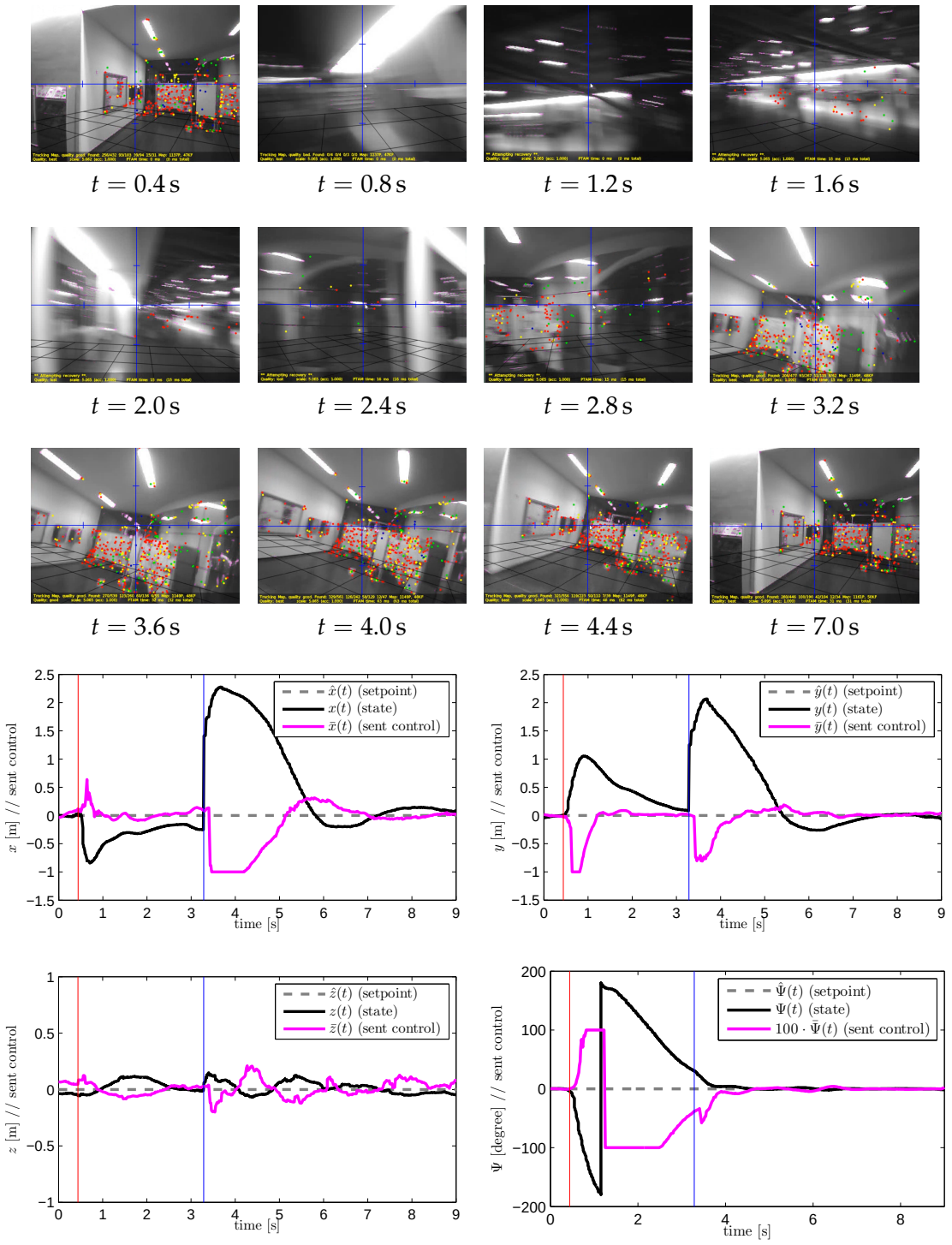


Figure 8.7. Top: the drone's video stream. Bottom: x (top-left), y (top-right) z (bottom-left) and Ψ (bottom-left) control. At $t = 0.5\,\mathrm{s}$, the drone is pushed and rotated away from its position, such that visual tracking gets lost (red vertical line). Using only IMU measurements, the drone attempts to return to its original position. At $t = 3.2\,\mathrm{s}$, visual tracking recovers (blue vertical line), the filter is updated with the visual pose estimates, and the drone can return to its real target position.

9. Conclusion

In this thesis, we developed a system that enables a low-cost quadrocopter - such as the Parrot AR.Drone - to localize and navigate autonomously in previously unknown and potentially GPS-denied environments. Our approach uses a monocular camera onboard the quadrocopter, and does not require artificial markers or external sensors.

To achieve this, our approach employs a keyframe-based, monocular simultaneous localization and mapping (SLAM) system to build a sparse map of the environment, and to compute a visual pose estimate within this map for each video frame. We contribute a novel method to estimate the scale of this map from inertial and altitude measurements by formulating the problem statistically, and deriving a closed-form solution for the maximum likelihood (ML) estimator of the unknown scaling factor.

We developed an extended Kalman filter (EKF) to fuse this visual pose estimate with additional sensor measurements available, to synchronize the different data sources and to compensate for delays arising from the communication process and the computations required for tracking a video frame. This is achieved by formulating and implementing a full model of the drone's flight dynamics and the effect of sent control commands, capable of compensating for delays of up to 400 ms.

Proportional-integral-differential (PID) control is applied to control the pose of the drone, and fly to and hold a desired target position: our approach allows the drone to approach its target position with a speed of up to $2\,\mathrm{m/s}$ and to accurately hold a flying position with a root mean squared error (RMSE) of only 8 cm.

On one hand, the system is robust with respect to temporary loss of visual tracking due to occlusions or quick rotation, as it is able to approximately return to its original pose using the IMU and altimeter measurements such that visual tracking can recover. On the other hand, the visual tracking eliminates drift, allowing the drone to navigate in a large area as long as the battery permits, and automatically land no more than 15 cm away from its takeoff position. We extensively tested the system in different real-world indoor and outdoor environments, demonstrating its reliability and accuracy in practice.

In summary, we showed in our experiments that a low-cost MAV can robustly and accurately be controlled and navigated in an unknown environment based only on a single monocular camera with a low resolution of 320×240 pixel, provided there are enough landmarks in its field of view. Our approach resolves encountered challenges such as estimating the unknown scale of the map or compensating for delays arising from wireless LAN communication. We showed how IMU-based odometry and visual SLAM can be combined in a consistent framework to balance out their respective weaknesses.

With the developed system, we contribute a complete solution that facilitates the use of

low-cost, robust and commercially available quadrocopters as platform for future robotics research by enabling them to autonomously navigate in previously unknown environments using only onboard sensors.

10. Future Work

There are a number of interesting research directions to augment and build upon the current system, which we will briefly discuss in this chapter.

Direct Incorporation of IMU Measurements and Increasing Map Size

One of the major limitations of the current system is the size of the visual map. The time required for tracking a video frame grows linearly with the number of landmarks in the map, severely limiting the map size in practice. There are several interesting approaches to enable constant-time tracking. For example enforcing euclidean consistency of the map only in a local window, while globally the map is treated topologically as recently proposed by Strasdat et al. [37]. Essentially this allows the map to grow arbitrarily without affecting the complexity of tracking a video frame or incorporating new keyframes.

An interesting topic for future research is how IMU and altimeter measurements - in particular with respect to the camera's attitude - can be integrated into this pose-graph framework as additional soft constraints on and in between consecutive keyframes. This additional information can not only be used to eliminate three out seven drift dimensions (scale, as well as roll and pitch angle), but can also be used as additional constraint between poorly connected or unconnected parts of the visual map. Consider flying through an open door or window: only very few landmarks will be visible from both sides of the door, causing visual tracking to be inaccurate or even fail completely. IMU measurements can then be used to impose relative constraints between keyframes taken on both sides of the door, allowing to continue visual tracking within the same map.

Increasing Robustness using Multiple Cameras

A further interesting research opportunity is given by the problem that when only one monocular camera is used, visual tracking is bound to fail if there are not enough landmarks in the field of view of that camera. This is particularly true for a low-resolution camera such as the one used for this thesis. An interesting topic for future research would be how incorporating multiple cameras can improve the tracking process by providing redundancy to e.g. a white wall in one direction.

Additionally, differently orientated cameras can provide strong redundancy to different types of motion: while tracking rotation around the optical axis of a camera is comparatively easy, large rotations around the x and y axis typically cause visual tracking to fail as the field of view changes rapidly, and - for monocular SLAM - new landmarks cannot be initialized due to the lack of sufficient parallax. Using three orthogonally orientated cameras, any type of motion can be tracked well by at least one of these cameras.

In this thesis we showed that accurate visual tracking can be done using a camera with a resolution of only 320×240 - in terms of computational complexity tracking four such cameras rigidly attached to each other (i.e. undergoing the same motion) is equivalent to tracking one 640×480 camera, as the tracking time only depends on the image size, the number of potentially visible landmarks and the number of landmarks found. Incorporating multiple low-resolution cameras pointing in different directions might hence provide significantly more accurate and robust tracking performance than using one high-resolution camera while requiring the same computational power. Due to the very low cost, weight and power consumption of such cameras, this is of particular interest in the context of miniature aerial vehicle navigation.

Onboard Computation

The current system depends on a ground station for computationally complex tasks such as visual SLAM, which is mainly due to the very limited access to the onboard processing capabilities of the Parrot AR.Drone. While this does allow for more computational power to be used, it is also the reason for some of the challenges encountered in the course of this thesis, in particular for the large delays due to wireless LAN communication.

A next step would be to make the drone fly truly autonomously by performing all computations required onboard - the comparatively low computational power required by the current system suggests that a similar approach can well be performed on integrated hardware. An interesting research topic would be how the approach presented can be divided in two parts, the time-critical tracking, filtering and controlling part running on integrated hardware while the computationally expensive map optimization is done on a ground station PC.

Such an approach would efficiently combine the benefit of visual tracking onboard (little delay) and the computational power of a ground station for optimizing and maintaining a large map. Additionally, little bandwidth would be required as only sparse representations of keyframes and landmarks need to be communicated, instead of streaming each video frame. This could be extended as far as to allow the drone to autonomously explore small new areas of the environment and then fly back in range of the ground station to drop new observations and keyframes, while receiving an optimized version of the global map.

Dense Mapping and Tracking

An interesting and very promising research topic is the transition from sparse, keypoint-based SLAM to dense methods, either using depth-measuring cameras or by inferring the three-dimensional structure of the observed scene from camera motion alone.

In particular an online generated, three-dimensional model of the environment greatly facilitates autonomous obstacle avoidance and recognition, path planning or identification of e.g. possible landing sites - after all the MAV is not supposed to fly into a white wall just because there are no keypoints on it. Furthermore, the capability of easily generating accurate, three-dimensional models of large indoor environments is of interest in industrial

applications - such a model can also be generated offline, after the MAV used to acquire the visual data has landed.

Due to the heavy dependency on GPU hardware of all dense SLAM methods, this would have to be combined with a hybrid approach, using a ground station for computationally complex tasks. The current development of GPUs for mobile phones however indicates that it will be possible to run such methods on integrated hardware in the near future.

Appendix

A. SO(3) Representations

Throughout this thesis, three different representations for three-dimensional rotations are used:

- as rotation matrix $\mathbf{R} \in \mathrm{SO}(3) \subset \mathbb{R}^{3\times3}$,
- as rotation vector $\mathbf{r} \in \mathrm{so}(3) \subset \mathbb{R}^3$,
- as roll angle ($\Phi \in \mathbb{R}$), pitch angle ($\Theta \in \mathbb{R}$) and yaw angle ($\Psi \in \mathbb{R}$).

While conversion between $\mathbf{r}$ and $\mathbf{R}$ is done using Rodrigues formula, there are different conventions regarding the interpretation of the roll, pitch and yaw representation, in particular with respect to multiplication order. For this work the following conversion formulae are used:

Roll, Pitch, Yaw to SO(3)

The roll (Φ), pitch (Θ) and yaw (Ψ) angles denote three consecutive rotations around different axis. The overall rotation matrix is given by:

$$R = \begin{pmatrix} \cos\Psi\cos\Phi & \cos\Psi\sin\Phi\sin\Theta + \sin\Psi\cos\Theta & \cos\Psi\sin\Phi\cos\Theta - \sin\Psi\sin\Theta \\ -\sin\Psi\cos\Phi & -\sin\Psi\sin\Phi\sin\Theta + \cos\Psi\cos\Theta & -\sin\Psi\sin\Phi\cos\Theta - \cos\Psi\sin\Theta \\ -\sin\Phi & \cos\Phi\sin\Theta & \cos\Phi\cos\Theta \end{pmatrix} \tag{A.1}$$

SO(3) to Roll, Pitch, Yaw

Given a rotation matrix $\mathbf{R}$, the corresponding angles can be extracted. Let

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \tag{A.2}$$

Roll, pitch and yaw angles can then be extracted by:

$$\begin{aligned} \Phi &= \mathrm{Atan2}\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right) \\ \Psi &= -\mathrm{Atan2}\left(\frac{r_{21}}{\cos(\Phi)}, \frac{r_{11}}{\cos(\Phi)}\right) \\ \Theta &= \mathrm{Atan2}\left(\frac{r_{32}}{\cos(\Phi)}, \frac{r_{33}}{\cos(\Phi)}\right) \end{aligned} \tag{A.3}$$

where $\mathrm{Atan2}(x, y)$ is defined by

$$\mathrm{Atan2}(x, y) := \begin{cases} \arctan \frac{y}{x} & \text{for } x > 0 \\ \arctan \frac{y}{x} + \pi & \text{for } x < 0,\ y \geq 0 \\ \arctan \frac{y}{x} - \pi & \text{for } x < 0,\ y < 0 \\ +\pi/2 & \text{for } x = 0,\ y > 0 \\ -\pi/2 & \text{for } x = 0,\ y < 0 \\ 0 & \text{for } x = 0,\ y = 0 \end{cases} \tag{A.4}$$

Bibliography

[1] M. Achtelik, M. Achtelik, S. Weiss, and R. Siegwart. Onboard IMU and monocular vision based control for MAVs in unknown in- and outdoor environments. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2011.

[2] ARDrone Flyers. AR.Drone — ARDrone-Flyers.com, 2011. [`http://www.ardrone-flyers.com/`].

[3] H. Bay, T. Tuytelaars, and L.V. Gool. SURF: Speeded-up robust features. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2008.

[4] C. Bills, J. Chen, and A. Saxena. Autonomous MAV flight in indoor environments using single image perspective cues. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2011.

[5] J. Canny. A computational approach to edge detection. *Conference on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679 – 698, 1986.

[6] H. Deng, W. Zhang, E. Mortensen, T. Dietterich, and L. Shapiro. Principal curvature-based region detector for object recognition. In *Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[7] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: Part I. *Robotics & Automation Magazine*, 13(2):99 – 110, 2006.

[8] E. Eade and T. Drummond. Edge landmarks in monocular SLAM. *Image and Vision Computing*, 27(5):588 – 596, 2009.

[9] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. of the Alvey Vision Conference*, 1988.

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

[11] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proc. of the 24th annual ACM symposium on User interface software and technology (UIST)*, 2011.

[12] S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Proc. of the International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, 1997.

[13] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365 – 1378, 2008.

[14] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007.

[15] L. Kneip, S. Weiss, and R. Siegwart. Deterministic initialization of metric state estimation filters for loosely-coupled monocular vision-inertial systems. In *Proc. of the International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[16] K. Konolige and M. Agrawal. FrameSLAM: From bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics*, 24(5):1066 – 1077, 2008.

[17] T. Krajník, V. Vonásek, D. Fišer, and J. Faigl. AR-drone as a platform for robotic research and education. In *Proc. of the Communications in Computer and Information Science (CCIS)*, 2011.

[18] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2011.

[19] V. Lepetit, F. Moreno-Noguer, and P. Fua. EPnP: An accurate $O(n)$ solution to the PnP problem. *International Journal of Computer Vision (IJCV)*, 81(2):155 – 166, 2009.

[20] H. Li and R. Hartley. Five-point motion estimation made easy. In *Proc. of the International Conference on Pattern Recognition (ICPR)*, 2006.

[21] T. Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision (IJCV)*, 30(2):79 – 116, 1998.

[22] D. Lowe. Object recognition from local scale-invariant features. In *Proc. of the International Conference on Computer Vision (ICCV)*, 1999.

[23] J. Matas, O. Chum, M.Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761 – 767, 2004.

[24] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid. A constant time efficient stereo SLAM system. In *Proc. of the British Machine Vision Conference (BMVC)*, 2009.

[25] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2011.

[26] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. In *Proc. of the International Symposium on Experimental Robotics (ISER)*, 2010.

[27] K. Mikolajczyk and C. Schmid. Scale & affine invariant interest point detectors. *International Journal of Computer Vision (IJCV)*, 60(1):63 – 86, 2004.

[28] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proc. of the AAAI National Conference on Artificial Intelligence*, 2002.

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

[30] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart. Fusion of IMU and vision for absolute scale estimation in monocular SLAM. *Journal of Intelligent Robotic Systems*, 61(1–4):287 – 299, 2010.

[31] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(3):448 – 461, 2010.

[32] Parrot. AR-Drone developer guide for SDK 1.6, 2011. [`http://projects.ardrone.org`].

[33] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2006.

[34] J. Shi and C. Tomasi. Good features to track. In *Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 1994.

[35] C. Stachniss. *Exploration and Mapping with Mobile Robots*. PhD thesis, Universiät Freiburg, 2006.

[36] H. Stewénius, C. Engels, and D. Nistér. Recent developments on direct relative orientation. *Journal of Photogrammetry and Remote Sensing (ISPRS)*, 60(4):284 – 294, 2006.

[37] H. Strasdat, A. Davison, J. Montiel, and K. Konolige. Double window optimisation for constant time visual SLAM. In *Proc. of the International Conference on Computer Vision (ICCV)*, 2011.

[38] H. Strasdat, J. Montiel, , and A. Davison. Scale-drift aware large scale monocular SLAM. In *Proc. of Robotics: Science and Sytems*, 2010.

[39] H. Strasdat, J. Montiel, and A. Davison. Real-time monocular SLAM: Why filter? In *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2010.

[40] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. MIT Press, 2005.

[41] Wikipedia. H.263 — Wikipedia, the free encyclopedia, 2011. [`http://en.wikipedia.org/w/index.php?title=H.263&oldid=439433519`].

[42] Wikipedia. Quadrotor — Wikipedia, the free encyclopedia, 2011. [`http://en.wikipedia.org/w/index.php?title=Quadrotor&oldid=443167665`].