Graz University of Technology

Institute for Computer Graphics and Vision

Master's Thesis

Human-Inspired Visual Servoing for Automatic Take-off, Hovering and Landing of MAVs

Mario Katusic

Graz, Austria, February 2012

Thesis supervisors

Univ-Prof. Dipl.-Ing. Dr. techn. Horst Bischof

Dipl.-Ing. Andreas Wendel

To every action there is always opposed an equal reaction.

Isaac Newton

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am …………………………… …………………………………………………..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…………………………… …………………………………………………..
date (signature)

Abstract

In this thesis we present a human-inspired visual servoing approach for automatic take-off, hovering, and landing of Micro Aerial Vehicles (*MAV*s), suitable for indoor and outdoor applications. Our approach is based on a Position-Based Visual Servoing (*PBVS*) technique. Therefore, we use only a monocular camera looking in flight direction as an input sensor. Based on a state-of-the-art monocular Simultaneous Localization And Mapping (*SLAM*) approach, we estimate the position of the *MAV* in the environment. For the initialization of the map, we extend the algorithm by exploiting an artificial marker to obtain the correct scale. Additionally, we incorporate a fuzzy logic design in order to get a robust position control without the need of a mathematical model of the *MAV*. We show that this controller tolerates noisy pose estimates without incorporating additional sensor measurements. In the experiments, we demonstrate that our approach achieves a performance comparable to several state-of-the-art approaches for hovering and during trajectory flights, but without the need for sensor fusion or specific mathematical models. Furthermore, we demonstrate that even low camera resolutions deliver a pose estimate which can be used for autonomous *MAV* navigation tasks. Finally, we discuss how to detect system failures and how to react in indoor as well as outdoor environments. Our approach is useful for a variety of *MAV* applications, including the autonomous inspection of power pylons where take-off, hovering, and landing of *MAV*s is essential.

Keywords: Visual Servoing, VSLAM, Fuzzy Logic, Micro Aerial Vehicle, Hovering, Take-off, Landing, Pose Estimation, Inspection.

Kurzfassung

In dieser Abschlussarbeit präsentieren wir einen vom Menschen inspirierten Visual Servoing Ansatz für das automatische Starten, Schweben und Landen von kleinen, unbemannten Flugobjekten, sogenannten *Micro Aerial Vehicles (MAVs).* Unser Ansatz kann sowohl in Räumen als auch draußen in freier Natur zum Einsatz kommen. Der Algorithmus beruht auf der *Position-Based Visual Servoing (PBVS)* Technik, bei der nur eine einzige Kamera als Input-Sensor verwendet wird. Mit Hilfe aufgenommener Bilder der Umgebung wird die Position des *MAV* anhand einer aktuellen bildbasierten *Simultaneous Localization And Mapping (SLAM)* Methode zur gleichzeitigen Lokalisierung und Kartenerstellung berechnet. Um eine metrische Skalierung zu erhalten, initialisieren wir die Kartenerstellung mittels einem künstlichen Marker. Des Weiteren haben wir einen Fuzzy-Logic-Regler entworfen, welcher kein mathematisches Modell des *MAV* voraussetzt und eine robuste Kontrolle der Position ermöglicht. Wir zeigen, dass dieser Positionsregler ohne zusätzliche Sensoren mit dem Rauschen der visuellen Positionsbestimmung umgehen kann. In den Experimenten demonstrieren wir, dass vergleichbare Ergebnisse zu aktuellen Forschungsansätzen beim Schweben und Abfliegen von Trajektorien erzielt werden, jedoch ohne Einbindung von zusätzlichen Sensoren oder spezieller mathematischer Modelle, die auf das *MAV* zugeschnitten sind. Zusätzlich zeigen wir, dass bereits mit einer geringen Kameraauflösung die Position bestimmt werden kann und damit eine autonome Navigation des *MAV* möglich ist. Außerdem erläutern wir, wie Systemfehler detektiert werden können und wie diese in Räumen und in freier Natur behandelt werden. Unser Ansatz ist für eine Vielzahl von Anwendungen nützlich, bei denen das Starten, Schweben und Landen eine wichtige Rolle spielen – beispielsweise bei der autonomem Inspektion von Hochspannungsmasten mit einem *MAV*.

Stichworte: Visual Servoing, VSLAM, Fuzzy-Logic-Regler, Unbemanntes Flugobjekt, Schweben, Starten, Landen, Posenbestimmung, Inspektion.

Acknowledgments

This master thesis would not have been possible without the support of many people. I am heartily thankful to my understanding and beloved parents, Ivka and Mijo, who gave me the moral and financial support I required during my studies and who always showed a great deal of patience. In addition, I would like to thank my sister Margareta and my girlfriend Adrijana Bandic, who supported me in any respect during the completion of the thesis.

I also would like to make a special reference to my dear friends Thomas Kempter and Michael Goller, many thanks for the numerous suggestions and the proofreading as well as the assistance during the evaluation of the experiments. Special thanks also to Dipl.-Ing. Michael Maurer for his advice and support with handling the hardware and his effort as a safety pilot.

Last but not least, I would like to express my gratitude to my supervisors, Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof and Dipl.-Ing. Andreas Wendel, who were abundantly helpful and offered invaluable assistance, support and guidance. I am very proud to have been able to work with such experts in computer vision.

Thank you!

The project is funded by the Austrian Research Promotion Agency (*FFG*) as FIT-IT project 825841/10397. In the year 2009 *PEGASUS* won the first prize in the category of *Visual Computing*.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

An Unmanned Aerial Vehicle (*UAV*) is an airborne machine or robot which is controlled remotely by a navigator or autonomously by preprogrammed flight plans. In recent years, autonomous *UAV*s have become a field of active research. Nowadays there is a wide variation of *UAV* shapes, sizes and characteristics. A *UAV* subclass is the Micro Aerial Vehicle (*MAV*), which is restricted in size and weight. For the autonomous flight of *UAV*s, automatic take-off and landing are critical phases. The **take-off** phase is defined as the start of the *UAV* from ground or from a starting platform until the aircraft reaches a certain height. The **landing** phase describes how the *UAV* gets back from the air onto the ground- or landing platform. In this case, it is of particular importance to touch the ground gently in order to avoid damage of the aircraft. Some types of *UAV*s, such as helicopters are capable of **hovering**, which means holding a constant position in the air. From a controller point of view hovering is a challenging task because of the unstable system dynamics.

1.1 Motivation

With *UAV*s becoming popular, a wide field of application has opened. *UAV*s are used not only in military applications but also for civil purposes, for instance when it is too dangerous for humans to perform a certain task. Scenarios where *UAV*s are employed include chemical accidents, firefighting missions and explosives defusing [1–3]. Another application is the field of inspection and surveillance, for example monitoring of oil and gas pipelines, large crowds or power lines and power pylons [4–6].

This thesis is part of the *PEGASUS* project at Graz University of Technologies which aims at the autonomous inspection of overhead power lines. Today, the inspection of overhead power lines is performed by an inspector from the ground if the power pylons are easily accessible. In this case the inspector thoroughly checks the high voltage insulators on the power pylon and inspects the power lines using binoculars. However, if the power pylons are placed remotely from infrastructure, the inspection is performed using helicopters as shown in Figure 1.1. The goal of the visual inspection is to find failures on the power lines and insulators such as sagging spans, broken or slack stay wires, broken or chipped insulators or discoloration due to corroded joints [6]. This process is very time consuming, dangerous, expensive and requires expert knowledge of the inspector.

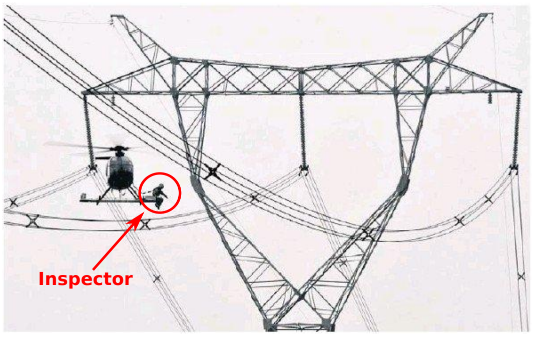


Figure 1.1: Traditionally power line and power pylon inspection is done by a helicopter.

The survey can also be done using camera equipped *UAV*s which also allow for an autonomous inspection based on computer vision algorithms. Compared to the traditional inspection process, this approach has several advantages. Since *UAV* flights are much more cost effective, an aerial vehicle can spend more time on the inspection of a particular power pylon. In contrast to traditional helicopters *UAV*s can fly closer to the object under inspection to perform a more sophisticated survey and documentation.

For automatic vision based inspection, several tasks are required [7]. First, continuous pose estimation is necessary to navigate the *UAV* during the inspection task. Second, object identification needs to be performed to find the object under inspection. Finally, when the desired object is found in the scene, specific algorithms are used to assess different object properties such as shape and surface condition. All of these tasks are performed using image processing and computer vision algorithms.

The benefits of computer vision systems are cost reduction, increased processing speed, reproducible results and the fact that expert knowledge is used as prior information and is no longer needed during inspection. Due to these advantages, the vision-based inspection with autonomous *UAV*s has the potential to replace the cost intensive conventional inspection. However, there are several challenges to overcome, like illumination changes, a wide variety of possible object shapes and sizes as well as the vast number of possible failures.

In general, all autonomous flight applications require sophisticated mechanisms for automatic take-off, hovering and landing, since these are the critical flight phases. This thesis presents a robust visual servoing system that can be used for the control of an autonomous *UAV* during these phases.

1.2 Project Goals and Contributions

The goal of our work is to develop a system for taking-off, hovering, navigating and landing an *MAV* solely based on visual input. The system needs to be capable of handling indoor and outdoor environments, it needs to operate based on natural features and it needs to be robust towards measurement and feature uncertainties.

Position Based Visual Servoing. The main contribution of this work is a Position-Based Visual Servoing (*PBVS*) system which allows a fully autonomous and robust control of an *MAV*, solely based on visual input data. The image sequences received from a single, monocular camera are used to determine the position in 3-dimensional space, and this position is used as an input for the control loop. The camera is mounted on the *MAV* looking forward rather than on the ground, which has the benefit that the captured images can also be used for inspection tasks.

Vision Based Pose Determination. An adapted state-of-the-art Visual Simultaneous Localization and Mapping (*VSLAM*) framework is used to generate a sparse, metrically scaled map of a local environment. This allows the *MAV* to localize itself in the region where the take-off and landing should be performed.

Human Inspired Position Control. The control of the *MAV* is inspired by the way a human being would control it. This has several advantages: First, there is no need for a mathematical description of the controlled system. Second, no model of the *MAV* physics is required. Finally, our approach yields a simple and intuitive control design with low implementation complexity and low computational costs.

Error Handling. In order to allow for a reliable operation the system implements mechanisms to deal with various error scenarios such as missing visual input data, pose estimation errors, or general hardware failures.

In the following, an overview of the field of *UAV* navigation is presented. This includes a brief review of the different possible sensor modalities with a focus on visual servoing systems. Furthermore, mechanisms for pose estimation are discussed and evaluated. Chapter 2 concludes with an introduction to control theory, focusing on fuzzy logic control. The system components (regarding hardware and software) are discussed in detail in Chapter 3. Chapter 4 presents a detailed description of the implemented mechanisms for pose estimation and control that are used for autonomous take-off, hovering and landing of the *MAV*. This chapter also discusses possible system failures and the suggested countermeasures. The implemented system is evaluated by a series of experiments, which are presented in Chapter 5, including simulation results as well as indoor- and outdoor flights. The thesis concludes with a summary and an outlook to possible future research in Chapter 6.

Chapter 2

Related Work

Contents

A variety of algorithms for perception and control on autonomous aerial vehicles have been proposed. These techniques mainly differ by the available and sensors which are used. We would like to differentiate between vision-based sensors, and sensors which do not use visual data. These include Global Positioning System (*GPS*) sensors, Inertial Measurement Unit (*IMU*) sensors, compasses, air pressure sensors, and ultrasonic sensors. Furthermore, different sensors can be fused so that several sensors are combined to a single *fused* measurement.

2.1 Sensing Techniques for Perception and Control

First, we give an overview of different autonomous flight approaches with different sensing techniques. In [8] the primary navigation sensor is *GPS*. The goal in this work is to make a *UAV* autonomously hover using a *GPS* receiver with four antennas. The system is able to estimate the position at a rate of 10 Hz, resulting in a position accuracy in the range of a few meters. However, *GPS* based systems tend to drift over time and the accuracy depends strongly on the number of available satellites. Moreover, an accuracy in the range of a few meters is not sufficient for the autonomous survey in the vicinity of obstacles. A combination of *GPS* and *IMU* is shown in [9] for an autonomous flight control system that is able to move along predefined waypoints. The *UAV* is controlled by a ground-station connected by a wireless link. The ground-station plans the flight trajectory and sends the *UAV* from one *GPS* waypoint to the next. The presented results show an acceptable accuracy in the range below one meter while hovering, but for an aircraft which operates in much higher altitudes where no obstacles can arise than what we aim for. In the case that the *UAV* flies near the ground more obstacles are present and a higher precision is necessary. However, the paper lacks a complete evaluation along a predefined trajectory.

Another possible sensor for autonomous flights is the Laser Range Finder (*LRF*) as used in [10–12]. There, the *LRF* is used for Simultaneous Localization and Mapping (*SLAM*). In addition to that, the *LRF* is also used as distance sensor for obstacle avoidance. In case of an obstacle in the planned path, an on-line path correction is performed. The *SLAM* approach has become very popular in the field of robotics, and there are several approaches that combine *LRF*s with other sensors such as odometry [13]. The drawback of *LRF* based *SLAM* approaches is that they do not allow a sophisticated object inspection and the time for scanning an entire volume in an outdoor environment is considerably higher than using cameras. Furthermore, long range *LRF*s contradict to the limited payload constraint of *UAV*s.

Combined color and depth cameras, Red-Green-Blue-Depth (*RGB-D*), have successfully been used to control a *UAV*'s altitude [14]. An *RGB-D* camera provides a color image, while for each pixel the depth is estimated using pattern projection of Infrared (*IR*) points. This sensor has become very popular and affordable, thanks to the *Microsoft Kinect* game controller. Huang et al. [15] show an autonomous *UAV* flight in an unknown environment based on such an *RGB-D* camera. The approach is additionally able to generate a detailed 3D reconstruction of the environment. However, the *RGB-D* camera cannot be used for outdoor application because of the *IR* sensor.

Most of the listed sensors either drift over time, yield unsatisfactory accuracy for *MAV* navigation or cannot be used in outdoor applications. For sensors with high accuracy, such as the *LRF*, the physical dimensions get rather big, which has the effect that they cannot be employed to *MAV*s because of a limited payload. From this point of view, vision-based systems have several advantages: They do not drift over time, are cheap, have small construction size and thus little weight and have low power consumption. Furthermore, a single sensor is able to detect motion in six Degrees of Freedom (*DoF*), and yields a higher precision compared to low-cost *GPS* sensors.

The advances in computer vision over the last years opened a wide field of applications where high accuracy is necessary. Considering the project goals, take-off, hovering and landing which are introduced in *Chapter* 1 and the advantages of vision based systems listed above, visual servoing is preferable for autonomous navigation of *UAV*s. The next section presents an more in-depth discussion of visual servoing systems.

2.2 Visual Servoing

Visual servoing techniques use visual input to control the motion of a robot [16, 17]. The input data can be obtained by a monocular or a multi camera system, mounted on a robot (eye-in-hand) or by a fixed system (eye-to-hand). Visual servoing is an interdisciplinary combination of computer vision and control theory. One of the first papers in the field of visual servoing was published by Hill and Park [18] in 1979. A more recent overview on this topic is given by Chaumette and Hutchinson [16].

From a mathematical point of view, visual servoing is a general error minimization problem [3] using visual input. The goal is to minimize the error

$$\boldsymbol{e}(t) = \boldsymbol{s}^* - \boldsymbol{s}(t), \tag{2.1}$$

where $\boldsymbol{s}^*$ is the desired system state and $\boldsymbol{s}(t)$ is a measured system quantity representing the current state of the system. Usually,

$$\boldsymbol{s}(t) = \boldsymbol{s}(\boldsymbol{m}(t), \boldsymbol{a}), \tag{2.2}$$

which means that the system state is obtained from image measurements $\boldsymbol{m}(t)$ and additionally depends on a set of parameters $\boldsymbol{a}$. The motion of the robot is controlled such that the deviation between the current and desired system state, $e(t)$ is minimized. In general there are two different types of visual servoing, namely Image-Based Visual Servoing (*IBVS*) and Position-Based Visual Servoing (*PBVS*) which are discussed in the following paragraphs.

2.2.1 Image-Based Visual Servoing

The *IBVS* concept directly uses 2D image information to control the motion of a robot. Therefore, $\boldsymbol{s}(t)$ is directly based on measurements $\boldsymbol{m}(t)$ taken from the image plane. For this reason $\boldsymbol{s}(t)$ can be interpreted as image features. The parameter set $\boldsymbol{a}$ consists of intrinsic parameters of the camera(s). The error $\boldsymbol{e}(t)$ can be expressed as the difference between the desired features $\boldsymbol{f_d}$ and currently extracted features $\boldsymbol{f_c(t)}$,

$$\boldsymbol{e(t)} = \boldsymbol{f_d} - \boldsymbol{f_c(t)}. \tag{2.3}$$

The block diagram in Figure 2.1 visualize this principle. The *IBVS* concept is a *teach by showing* technique, which means that a target image taken from the desired position is required. Using this image, features are extracted and compared to the current image features to compute the error $\boldsymbol{e}(t)$ that serves as controller input.

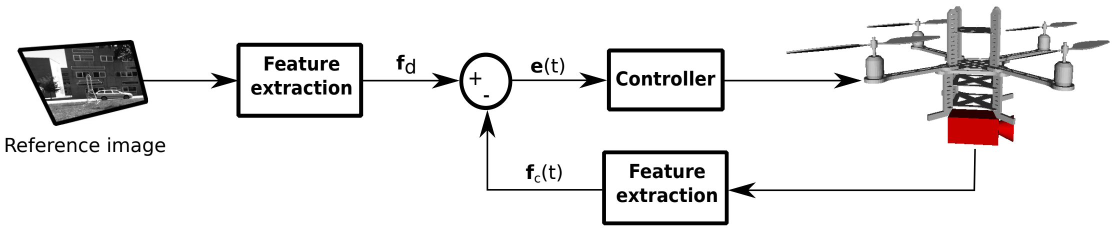


Figure 2.1: The *IBVS* concept. Features extracted from every taken image are continuously compared to the features of the reference image on the left-hand side. The difference $e(t)$ serves as input to the controller.

The motion control of the robot is done in such a way that the current features in the image plane attain the position of the desired features. For this reason, it is necessary to describe how the image features $\boldsymbol{f}(\boldsymbol{x})$ change when the position and orientation $\boldsymbol{x}$ of the camera changes. This description is given by the Jacobian matrix $\boldsymbol{J}$ which is defined as

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \frac{\partial f_1(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\boldsymbol{x})}{\partial x_l} \\ \frac{\partial f_2(\boldsymbol{x})}{\partial x_1} & \frac{\partial f_2(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\boldsymbol{x})}{\partial x_l} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_k(\boldsymbol{x})}{\partial x_1} & \frac{\partial f_k(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f_k(\boldsymbol{x})}{\partial x_l} \end{bmatrix}. \tag{2.4}$$

This matrix consists of k features $f_1 \dots f_k$ (number of rows) and l velocity parameters $x_1 \dots x_l$ (number of columns). Using the Jacobian matrix the system dynamics can be written as

$$\frac{\boldsymbol{df}}{dt} = \boldsymbol{J}\dot{\boldsymbol{x}} = \boldsymbol{J}\boldsymbol{v}, \tag{2.5}$$

where $\boldsymbol{v}$ denotes the robot's velocity.

In recent years, *IBVS* approaches have become popular in autonomous *UAV* applications. Since the approaches differ in type of aircraft, sensor used modalities, and image processing techniques, a comparison of different systems is difficult. Moreover, there is no standardized way for comparing system accuracy.

Azinheira and Rives [19] present a system for an automatic flight along predefined trajectories, which are indicated by lines on ground. The image features are selected in a way that enables a decoupling of rotational and translational control. This allows to separate lateral and longitudinal motion of the *UAV*. The evaluation of the approach is performed in a simulation framework and yields an accuracy in the range of several meters. However, no outdoor results with real-world data are presented. Mejias et al. [4] present a real-time system for *UAV* control using the *IBVS* approach in urban areas. The initialization is performed from a ground station by manual feature selection while the helicopter is hovering in front of a building. This approach uses the Lucas-Kanade tracker [20, 21] as a real-time feature tracking algorithm, which provides a certain robustness against uncertainties in the captured images.

In [22], Goncalves et al. use Euclidean homographies computed between subsequent images for controlling an aircraft in the approach and landing phase. This system employs trajectory planing in the image space and performs an interpolation between subsequent key-frames. The approach is validated using a simulator environment; however, no real-world experiments were performed.

The major advantage of the *IBVS* concept is its robustness to coarse camera calibration as described in [16]. Additionally, no object or scene models are required. The major disadvantage is that a reference image is required, which makes this approach unsuitable for unknown environments. The *IBVS* approach could be used for a *UAV* during the actual inspection of an object, where it is necessary to hold the current positions. However, *IBVS* does not allow navigation in unknown environments, since a reference image is required.

2.2.2 Position-Based Visual Servoing

In contrast to *IBVS*, the *PBVS* concept defines the error $\boldsymbol{e}(t)$ as

$$\boldsymbol{e}(t) = \boldsymbol{F}_d - \boldsymbol{F}_c(t), \tag{2.6}$$

where $\boldsymbol{F}_d$ is the desired pose and $\boldsymbol{F}_c$ the current pose of the robot. The pose estimation is obtained from a function of image measurements $\boldsymbol{m}(t)$, and additionally from parameters $\boldsymbol{a}$ which denote the intrinsic camera parameters, as well as information of a map or a single 3D-model in the scene, for instance the size of a Augmented Reality (AR) marker or a Computer-Aided Design (CAD)-model. The pose thus can be written in matrix notation as

$$\begin{aligned}
\boldsymbol{F}_c &= \begin{bmatrix} \boldsymbol{t}_c \\ \theta \boldsymbol{u} \end{bmatrix}, \\
\boldsymbol{F}_d &= \begin{bmatrix} \boldsymbol{t}_d \\ 0 \end{bmatrix}, \text{ and} \\
\boldsymbol{e} &= \begin{bmatrix} \boldsymbol{t}_d - \boldsymbol{t}_c \\ -\theta \boldsymbol{u} \end{bmatrix},
\end{aligned} \tag{2.7}$$

where both poses consist of a translation vector $\boldsymbol{t}$ and the current pose $\boldsymbol{F}_c$ contains additionally a rotation angle θ and the rotation axis $\boldsymbol{u}$ [16].

As shown in Figure 2.2, $PBVS$ requires two poses, namely the current pose $\boldsymbol{F}_c$ and the desired pose $\boldsymbol{F}_d$. The desired pose and the current pose of the robot are measured in one and the same coordinate frame. In general, the desired pose of the robot is the reference for the control law. The 3D pose of the robot is estimated in every frame and represents the feedback information for the controller.

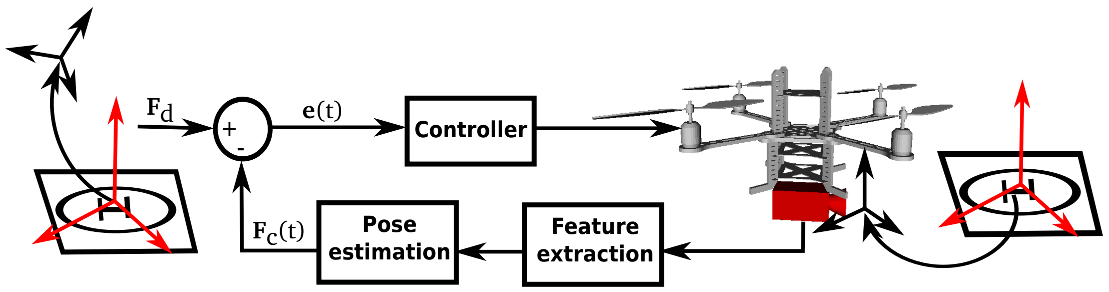


Figure 2.2: The $PBVS$ concept. On the left-hand side the reference pose $\boldsymbol{F}_d$ for the controller is defined. Features are extracted and the current pose $\boldsymbol{F}_c(t)$ is estimated for every image taken. The error $\boldsymbol{e}(t)$, the difference of the two poses, serves as input for the controller.

The error from Equation (2.7) is minimized for instance using a single proportional controller defined by the matrix $\boldsymbol{K}$, so the system dynamics result to

$$\dot{x} = \boldsymbol{K}\boldsymbol{e}. \tag{2.8}$$

Thus, the current pose of the robot is propagated towards the desired pose. The error $e(t)$ of the current pose as well as the desired pose can be calculated in the Cartesian coordinate frame, which makes it easier to plan a trajectory for the robot.

The major advantage of *PBVS* in contrast to *IBVS* is that it does not require images representing a goal position. A disadvantage, however, is that *PBVS* is very sensitive to a good camera calibration as well as object parametrization. Using a badly calibrated camera will result in poor pose estimation results and furthermore a performance decrease of the visual servoing system.

In [23], Sharp et al. show a *PBVS* approach for automatic landing of a *UAV*. A pose estimate is provided using a known landing target. The algorithm operates at 30 Hz, which ensures real-time capability during autonomous flights. The main part of the work deals with segmentation and feature extraction of the landing target, while the features are labeled and pose and speed are estimated with reference to the target. Pose estimation is accurate up to an RMS error of about 5 cm in translation and 5 degrees in rotation. The authors do not discuss the vision based control and no autonomous flight experiments are conducted.
Another paper by Saripalli et al. [2] shows a real-time vision based landing algorithm for helicopters and navigation from an initial pose to a final pose in partially known environments. The authors also use a landing target for pose estimation. When the target is not within the camera's view, *GPS*-based control is used as a backup system. The approach claims to be a robust method for autonomous landing within the translational precision of 40 cm distance to the landing target and 7 degrees rotation variation.
Teulière et al. [5] present a model-based vision system featuring position control for quadrotor helicopter in indoor environments using a *PBVS* approach. The camera is mounted on a *UAV* looking downwards and the visual data is fused with data from an *IMU*. The fusion is performed by an Extended Kalman Filter (*EKF*) [24]. The entire calculation runs on a ground-station, and is split into three parts, including the control scheme of the *UAV*, visual tracking of the known 3D-models for position estimation, and the fusion of the vision- and *IMU*- data. The authors demonstrate the robustness of the position controller for quadrotor helicopters by hovering in indoor environments within a precision of 15 cm on the x- and y-axis, and 30 cm on the z-axis.
Target or model based approaches have the major disadvantage that a visual target is required for autonomous flights. The landing task can be done provided that the target is present and visible to the *UAV*, which is not the case in unknown environments.

One of the first *PBVS* techniques in unstructured environments has been presented by Blösch et al. [1]. The pose estimation works without any artificial markers, and is based on a state-of-the-art *VSLAM* algorithm [25]. For the control task, the Linear Quadratic Gaussian Control Design with Loop Transfer Recovery (*LQG/LTR*) is used. The *LQG/LTR* controller provides stabilization of current and desired pose, take-off, hovering, way-points following and autonomous landing. The camera is mounted on a *UAV*, looking downwards, and is directly connected to a ground-station by cable. In this approach the visual data is fused with the *IMU*-data to gain additional sensor information. The approach is robust and shows high precision; when hovering, the maximum absolute error is 11.15 cm.
Achtelik et al. [26] present an algorithm for onboard vision-based *UAV* control for unknown in- and outdoor environments. The approach does not need any artificial markers, nor information about the environment in terms of model parameters. Like Blösch et al. [1], the authors employ the current state-of-the art framework, Parallel Tracking and Mapping (*PTAM*) [25], for *VSLAM*. With the help of accelerometers and the air pressure sensor of the *UAV*, the absolute scale is estimated to generate a correctly scaled map. A benefit of the approach is that the entire computation is performed on-board the *UAV* without the need for a ground-station.
Despite the accurate results, these two systems also show disadvantages. First, a detailed mathematical model is needed for the design of the *LQG/LTR* controller. These models are quite specific to the type of *UAV* and need precise parameter configuration. The second disadvantage is that the camera is pointing downwards, meaning that it can only be used for navigation and not for inspection tasks.

The previous discussion of the *PBVS* approach implies that this approach is suitable for controlling a *UAV* during autonomous flights in unknown environments. One aspect which is common to all discussed *PBVS* approaches is the need for an accurate pose estimation. The desired and current pose are necessary to be able to control the motion of a robot. Therefore, the quality of the control mainly depends on the accuracy of pose estimation. Hence, the next section deals with visual pose estimation in detail.

2.3 Visual Pose Estimation

During visual pose estimation a rigid transformation consisting of a rotation $\boldsymbol{R}$ and a translation $\boldsymbol{t}$ of the camera with respect to a fixed coordinate frame is calculated, as shown in Figure 2.3.
In the computer vision literature, pose estimation is often called extrinsic calibration [27, 28]. The pose estimation needs a minimum of three known correspondences between the 2D-image, and the

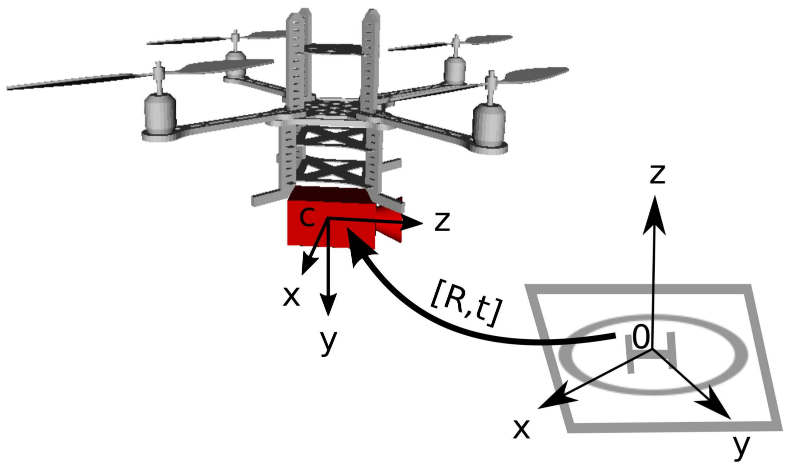


Figure 2.3: The pose estimation yields the rigid transformation, the 3D rotation $\boldsymbol{R}$ and the 3D translation $\boldsymbol{t}$, of the camera with respect to an object coordinate system.

3D-object, and is called the Perspective-3-Point-Problem (*P3P*) [29]. In a mathematical notation, the pose $\boldsymbol{R}$, $\boldsymbol{t}$ together with the intrinsic camera calibration matrix $\boldsymbol{K}$ form the projection matrix

$$\boldsymbol{P} = \boldsymbol{K}\left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]. \tag{2.9}$$

The projection matrix $\boldsymbol{P}$ is used to project a 3D-world point $\boldsymbol{X}$ to its corresponding 2D-image point $\boldsymbol{x}$ as

$$\boldsymbol{x} = \boldsymbol{P}\boldsymbol{X}. \tag{2.10}$$

Therefore, to extract a pose from an image, two cases have to be distinguished. The first is used for uncalibrated cameras, where the projection matrix $\boldsymbol{P}$ has eleven *DoF* and a minimum of six points are necessary for the pose estimation. Therefore,

- the pose (rotation $\boldsymbol{R}$ with three *DoF*, and translation $\boldsymbol{t}$ with three *DoF*) and
- the calibration matrix $\boldsymbol{K}$, which consists of the intrinsic parameters (focal length in x and y direction, principle point offset in x and y direction, and a skew factor)

are estimated at same time.

The second technique is used for calibrated cameras where $\boldsymbol{K}$ is known a priori. Thus, only the rotation $\boldsymbol{R}$ and the translation $\boldsymbol{t}$ of the camera are delivered as output. The basis of this approach is the known scene geometry, where the angle between pairs of 2D-points need to be the same as the angle between the corresponding 3D-points. Further details and different approaches on pose estimation can be found in [27, 28].

We now want to differentiate between pose estimation using a known object, such as artificial markers and *VSLAM* approaches which use natural features.

2.3.1 Artificial Markers

Artificial markers are target objects with simple, yet known geometry. When using such markers it is easy to distinguish the target object from other objects, even in complex environments. If the complexity of the environment grows, more time is needed to estimate the target's position. In *AR* applications the artificial markers are widely used to determine the position of the camera [30–32]. *AR* means that the real world environment is extended with artificial, overlaid information. Within an image or video, virtual objects or additional text information can be added as shown in Figure 2.4. Therefore, it is necessary to compute the camera pose and render the virtual object into the image in real time.

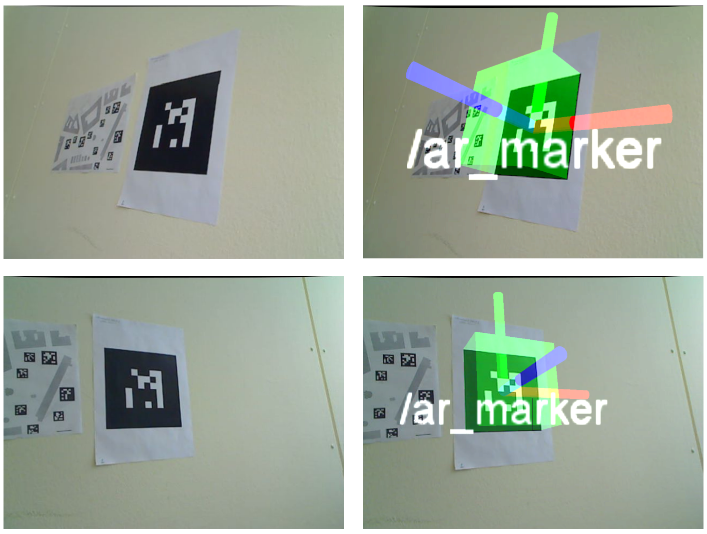


Figure 2.4: *AR* system with artificial markers. On the left-hand side the camera image is shown and on the right-hand side the augmented view. The view is extended with text and virtual objects, such as a simple shape and a coordinate frame. The virtual object can be rendered from different views based on the estimated pose.

One of the first works using 2D markers based on a squared-shaped barcode, which was also capable of detecting a large number of markers simultaneously, has been introduced by Rekimoto [30]. However, artificial markers just got popular with the work of Kato et al. [31]. As an outcome of this work the ARToolKit framework was presented, which is related to the approach of Rekimoto.

Nowadays, a lot of libraries for six *DoF* camera tracking with artificial markers exist. Wagner and Schmalstieg presented the ARToolKitPlus [32] library, which is optimized and extends the possible use to devices with limited computational power such as mobile phones. ARToolKitPlus has several advantages, including that it can be used on different platforms, that it uses heuristic automatic thresholding which can deal with illumination changes, and that it improves planar pose estimation using the *Robust Planar Pose Tracking* algorithm by Schweighofer and Pinz [33].

The main disadvantage of artificial markers is that the markers have to be placed in the environment. In cases of large environments or outdoor applications it is not always possible to place markers in such a way that a pose can be estimated at all times. Therefore, the *SLAM* technique which does not need artificial markers for pose estimation is discussed in the following.

2.3.2 Visual Simultaneous Localization and Mapping

For large environments, several approaches use a *SLAM* technique to track the robot's position [13, 34, 35]. This technique has to answer the following two questions:

- Where am I? Tries to determine the current position of the robot in a map.
- What does the world look like? Tries to create a map from the structure of the environment.

SLAM tries to estimate both, the robot's location and the structure of the environment in form of a map simultaneously, which is a chicken-and-egg problem. On the one hand a precise map is required for good robot localization, but on the other hand for a good map generation accurate localization is needed. Therefore, the method tries to estimate both at the same time. Either the sensor delivers 3D-measurement data directly, like *RGB-D* or *LRF*s, or the 3D-data has to be generated from several sensor measurements. For instance, when using a single camera, 3D-world points have to be to triangulated from a pair of suitable stereo images.
If a camera is used as input sensor, then the approach is called visual *SLAM* (*VSLAM*). Such a system mainly takes images as input, and provides the robot's pose and an abstract world representation as output. Often probabilistic approaches are used because of the noisy measurements of the sensors and the possible error accumulation over time. The environment is represented by features such as edges, corners, and/or planes. The current pose update is always based on the previous position of the robot.

Probabilistic approaches. Kalman Filter [36] and Particle Filter [37] are commonly used as probabilistic approaches to solve the *SLAM* problem. One of the first real-time monocular *VSLAM* approaches was presented by Davison et al. [38], who used a Kalman Filter for the localization and mapping task. This approach has the advantage that it is simple to implement and works well in practice under the assumption of linear Gaussian noise. The disadvantage is that a linear motion model is required and that the approach is limited to Gaussian probability distributions.
Since real processes almost never have a Gaussian probability distribution, a Particle Filters can be used to trace multiple hypotheses simultaneously. The probabilistic distribution function is approximated with a set of samples, which are weighted by their likelihood. Every particle represents a hypothesis about the position of the robot. The weight of a particle is a quality indicator for the probability of the state. At the beginning, each particle has the same weight and particles are uniformly distributed over the map. The particle weights are continuously updated as more information from sensor readings becomes available, and are propagated to a new location using a motion model. Based on the weights, a re-sampling of the particles is performed, which means that particles with small weight are removed and particles with higher weight are duplicated. The disadvantage of the approach is the growing computational complexity with the number of particles. Furthermore, it is hard to define the optimum number of particles for a particular system.

A Particle Filter-based approach for the *SLAM* problem, called *FastSLAM* is suggested by Montemerlo et al. [34, 35]. *FastSLAM* recursively estimates the pose of the robot and landmarks by taking the newest sensor measurements into consideration. Compared to the standard Kalman Filter, the suggested implementation has a computational complexity of $O(M \log(K))$, where M denotes the number of particles and K is the number of landmarks. The authors improved the convergence and accuracy of their approach in cases of linear *SLAM* problems.

Keyframe-Based approaches. In contrast to the probabilistic approaches described above, keyframe-based systems require visual input data obtained by a camera and do not estimate the pose according to probabilistic considerations. Given the visual input data, features are extracted to generate a map of the environment. Keyframes contain the camera position estimate as well as the extracted feature points, and are stored whenever the map needs to be extended. The pose estimate is continuously refined over several keyframes by minimizing the backprojection error of the extracted feature points according to a cost function. A current state-of-the-art implementation of a keyframe-based pose estimation from monocular images is the Parallel

Tracking and Mapping (*PTAM*) approach by Klein and Murray [25]. Since a modified *PTAM* implementation is used in this work, we discuss the working principle in more detail in Chapter 4.

In this section we have given an overview of methods for pose estimation with a focus on vision-based approaches. We have presented different methods and pointed out the advantages and disadvantages of each approach. The second major part of *PBVS* concepts is the control of a mobile robot. Therefore, different approaches in controller design are presented in the next section.

2.4 Control Theory

Control theory is a large area of research, thus we only give a short introduction covering material necessary to understand our work. Detailed information is given in [39–41]. To control a robot means to influence its dynamic behavior in a way to reach a desired goal. The desired goal in control theory is called reference r. In general, there are two ways to modify the dynamic behavior of a system. The first option, called "Open-loop control" excites the system with a certain input signal to obtain the desired output without measuring the current output signal. In contrast to that, "Feedback control" systems measure the current system output and adjust the reference signal accordingly. In more detail, feedback controllers use the measured system output y to calculate the difference e to the reference input r. The controller then takes the error e as input and delivers the system input u to influence the system dynamic, such that the error e is decreasing, as shown in Figure 2.5.

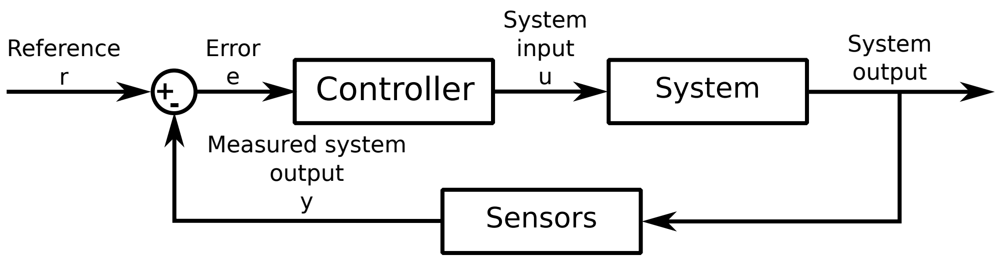


Figure 2.5: The feedback controller. The controller input is the error e, which is the difference between the reference r and the measured system output y. The output of the controller is the system input u to influence the system dynamics. The sensors measures the system state and close the feedback-loop.

Feedback control has several advantages in comparison to the open-loop control, including that unstable system can be controlled, unmeasured influences like friction or specific air resistance can be handled, and it is not sensitive to model parameter drifts.

In control theory it is necessary to define the *system* in mathematical terms. The system characteristics describe the physical quantity of a complex process, such as the temperature behavior of a heated room. To control the temperature of a room, a mathematical model of the thermodynamics of the room is needed. In practice it is often hard to describe the physical relations of a process, as parameters are often unknown and have to be described mathematically based on the laws of physics or can be estimated experimentally. The dynamics of a system are usually represented using differential equations. The problem is that real processes are predominantly partial and nonlinear and cannot be solved with standard linear control theory tools. The basic principles in control theory can only operate with linear differential equations. Therefore, the most common representation of a mathematical model is the state-space formulation:

$$\begin{aligned} \frac{d\boldsymbol{x}}{dt} &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}u, \\ y &= \boldsymbol{c}^T\boldsymbol{x} + du, \end{aligned} \tag{2.11}$$

where u is the system input and y the system output. Furthermore, $\boldsymbol{x}$ is the state vector, $\boldsymbol{A}$ the system matrix, $\boldsymbol{b}$ describes how the input influences the system states, $\boldsymbol{c}^T$ describes how the system output depends on the system states, and d represents the direct connection of the input signal to the output signal.

If no mathematical description is available, experimental system identification is used to identify the system's parameters. It measures the system output y based on the system input u and tries to estimate the transfer function of the system. The transfer function describes the mathematical relationship between input and output. Therefore, test functions such as a step-, impulse-, ramp- or sin-function are used as a input, and the response of the system output y is measured as shown in Figure 2.6. Depending on the output the system can be categorized as shown in Figure 2.6 into:

- **P-system:** A stepwise change of the system input u produces a stepwise change of the system output y. The transfer function thus has a proportional characteristic.
- **I-system:** A stepwise change of the system input u produce a linear change of the system output y over time. Thus, this transfer function has an integral characteristic.
- **D-system:** A stepwise change of the system input u produces a peak change of the system output y. Therefore, this transfer function has a differential characteristic.

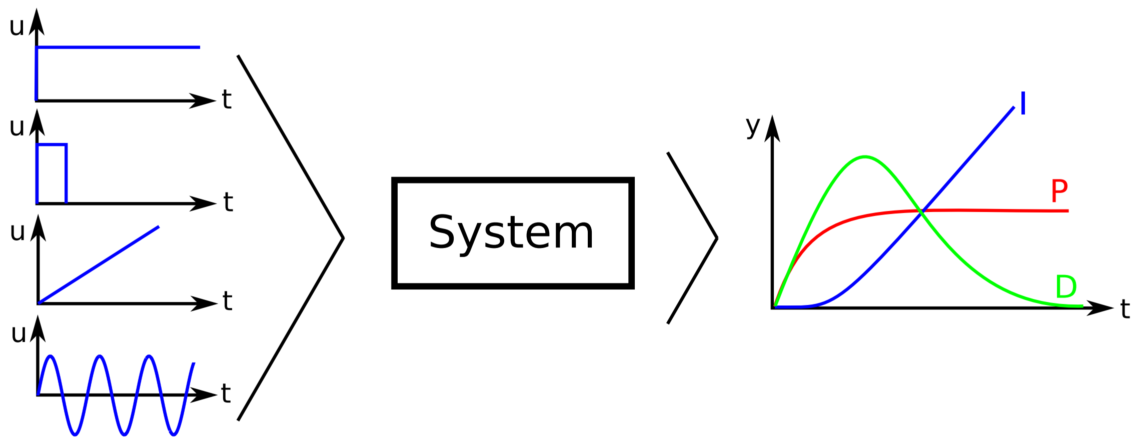


Figure 2.6: System identification: To identify a system, it is stressed with different input test functions, such as a step-, impulse-, ramp- or sin-function. By evaluating the corresponding output signal, the system characteristics can be estimated.

The stability of a control system is very important, since instability results in unpredictable system behavior which can have serious practical consequences. A system can be stabilized by means of a properly designed controller. According to the Bounded Input Bounded Output (*BIBO*) criterion, a system is stable if a bounded input produces a bounded output [39]. Another stability criterion has been defined by Lyapunov [41]. The Lyapunov stability analyzes the stability of a system near an equilibrium point $\boldsymbol{x}_R$ of the system state. If the dynamical system starts $\boldsymbol{x}_0$ near an equilibrium point and stays near the point $\boldsymbol{x}_R$ over time, then the system is Lyapunov-stable. The criterion is best illustrated graphically, as shown in Figure 2.7.

In mathematical terms,

$$\|\boldsymbol{x}(0) - \boldsymbol{x}_R\| < \delta \tag{2.12}$$

$$\|\boldsymbol{x}(t) - \boldsymbol{x}_R\| < \epsilon, \ \forall t \tag{2.13}$$

$$\lim_{t\to\infty} \|\boldsymbol{x}(t) - \boldsymbol{x}_R\| = 0. \tag{2.14}$$

If Equation (2.12) and (2.13) hold, the system is considered Lyapunov stable and if the harder constraint Equation (2.14) is fulfilled, the system is asymptotically stable.

So far, we have defined a system and important characteristics in terms of control theory, which is the basis for successful control such a system.

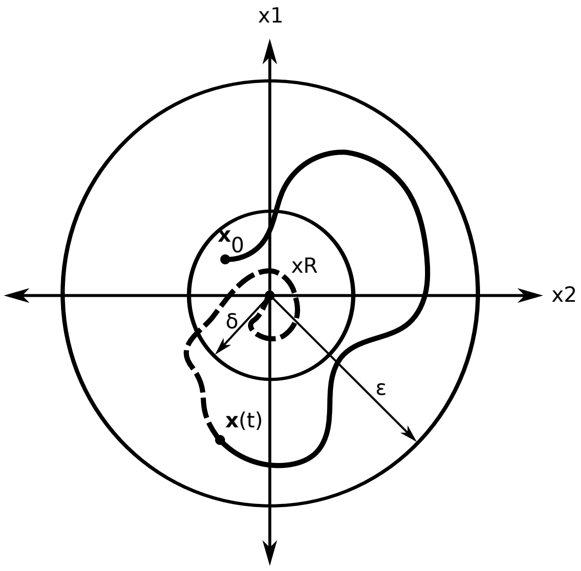


Figure 2.7: A system is defined as stable by the Lyapunov criterion [41], if the start point $\boldsymbol{x}_0$ lies within a distance δ, and over time never gets outside of ϵ within the state-space. If and only if the state-space variable $\boldsymbol{x}$ returns to the equilibrium $\boldsymbol{x}_R$ with $t \to \infty$ the system can be considered asymptotic stable, as depicted by the dotted line.

2.4.1 PID Controller

The most popular and widely used controller in industrial applications is the proportional, integral and differential (*PID*) controller. For a *PID* controller, the relationship between the input signal (the error e) and the output (the system input u) is

$$u(t) = K_P\, e(t) + K_I \int_0^t e(\tau)\,\mathrm{d}\tau + K_D \frac{\mathrm{d}e(t)}{\mathrm{d}t}, \tag{2.15}$$

where K_P is the proportional gain applied to the current error, K_I is the integral gain applied to the control error over time, and K_D is the differential gain applied to the derivative of the error. Figure 2.8 depicts the control loop of a *PID* controller, where the proportional-, integral- and differential factors operate in parallel. In the remainder, the single coefficients of equation (2.15) are discussed in more detail.

Proportional gain $\mathbf{K_P}$. The proportional gain K_P changes the output u in relation to the error e in form of a constant multiplication. If K_P is too high it could have a bad effect on the stability of the system: If the control error is high then the change of the output would also be very high, and the system could turn unstable. The other way round, if the K_P term is too small, then a high

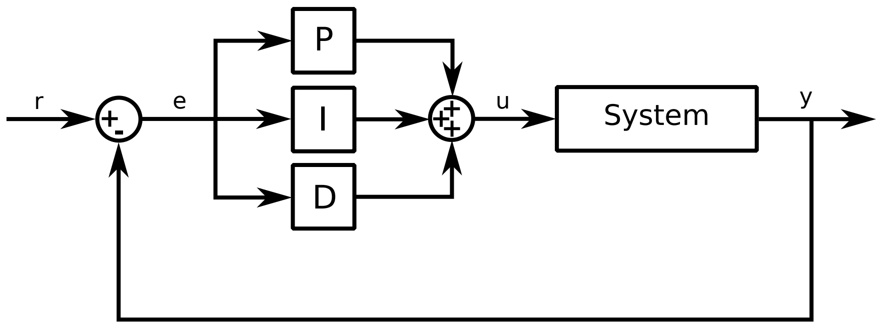


Figure 2.8: Classical structure of a PID controller.

error influences the system for a long time and thus the controller's sensitivity is too small. A disturbance to the system would permanently increase the control error.

Integral gain $\mathbf{K_I}$. The integral gain K_I changes the output of the controller such that the accumulated error over time is multiplied by a constant value K_I. This gain has the benefit that the error seen over a long time is going towards zero. Therefore, no steady-state errors occur, in contrast to system which only use a proportional term. The integral gain, however, can produce an overshoot of the system output if the value is too high.

Differential gain $\mathbf{K_D}$. The differential gain K_D is used to react on fast changes of the error over time. Furthermore, it can be used to reduce the overshoot from the integral gain, however, the differential gain is very sensitive to noise. Large amounts of noise and a high K_D value can make the control system unstable as well.

There are different ways to estimate the gain factors K_P, K_I, and K_D of the *PID* controller [39, 40], which are not discussed in details here.

The *PID* control design has been recently used for *UAV* control. In [42], a *PID* controller is used to stabilize an unstable flying robot on the basis of additional sensors, including accelerometers, a compass sensor and gyro sensors. The *PID* controller is only used for attitude control, which increases the control comfort for a human operator. Others use a *PID* controller to control the altitude and attitude of a *UAV* [43]. In this work, the *UAV* is able to hover in an indoor environment, whereas the pose of the *UAV* is determined by an outside-in tracking system. In [44], a *IBVS* approach is presented, where a *UAV* without additional sensors can hover in an indoor environment using a *PID* controller.

Employing a *PID* controller for controlling a *UAV* has the advantage of a relatively simple controller design, with the disadvantage that the control design cannot be done without explicit knowledge about the system. If the system would be known, there exist a variety of methods which require a lot of intuition and expert knowledge. Moreover, *PID* controllers are not suitable for nonlinear systems.
As a common alternative in controller design which does not require a mathematical model of the plant and can cope with system nonlinearities, the fuzzy control design in discussed in the following.

2.4.2 Fuzzy Logic Controller

Another control strategy which is very different compared to the classical *PID* control is called fuzzy logic control. Fuzzy logic began with the approach of the *fuzzy set theory* by Zadeh [45] and found a wide field of applications in control theory. As already mentioned it is often hard to describe the relationship between complex processes mathematically. Therefore, Zadeh developed a tool which can handle a complex problem using uncertain, imprecise reasoning, just like human beings do. The design of the controller is performed in a linguistic way, similar to the way a human would formulate rules. For example, the process of hovering a *UAV* can be described as follows: If the *UAV* is falling, then increase the thrust, and if the *UAV* is rising, then lower the thrust. The process is executed until the *UAV* is hovering. In this case no mathematical model of the *UAV* is necessary to hover. A human, cannot measure the height of the *UAV* exactly, but applying fuzziness he is able to make a *UAV* hover. In the literature, [3, 28, 46], a fuzzy logic controller is divided into three main parts, namely fuzzification, inference and defuzzification, as shown in Figure 2.9.

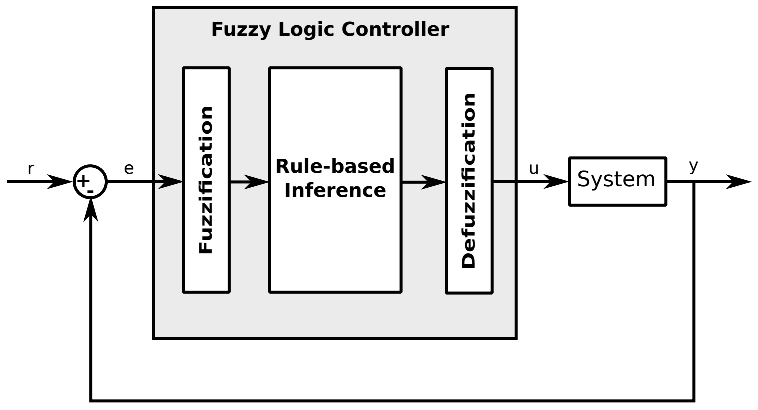


Figure 2.9: Block diagram of a fuzzy logic controller, with main parts bing the fuzzification, inference and defuzzification.

Fuzzification. The fuzzification does the mapping between the exactly measured values and the truth values of the fuzzy set. The fuzzy set is defined in form of membership functions, which give the trueness of a linguistic variable as depicted in Figure 2.10. When looking at the hovering example, there are three membership functions, namely *FALLING*, *HOVERING* and *RISING*. The membership functions are defined as triangles or trapezoids because of the easy linear mathematical description in contrast to nonlinear functions like Gaussian- and Sigmoid-functions.

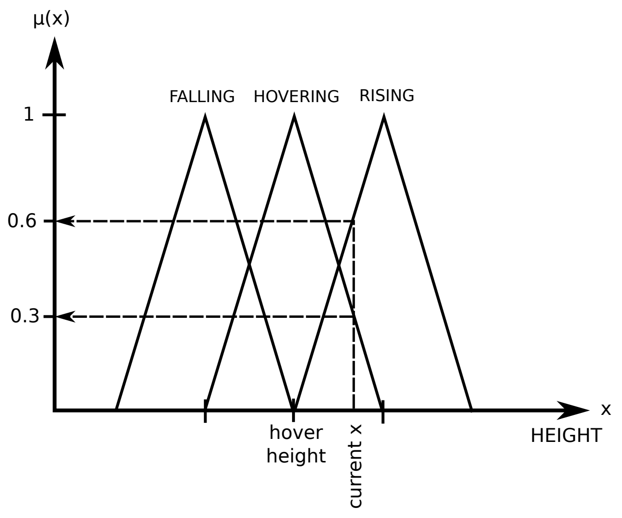


Figure 2.10: The fuzzification process. Given the current height the truth values of the memership functions are evaluted.

The mathematical description of fuzzy set A is

$$A = \{ (x, \mu_A(x)) | \, x \in X \}, \tag{2.16}$$

where x is a sensor measurement or system state, X the whole set of sensor measurements or system states, and $\mu_A(x)$ is the truth value of a membership function of the fuzzy set. The membership function is usually normalized between zero and one. An example of a fuzzification is given in Figure 2.10, where the current *UAV* height x has a truth value of 0.3 for the *HOVERING* and a truth value of 0.6 for the *RISING* membership function.

Inference. The inference part contains the rule set. The rules are defined in the same way a human expert would control the process. Therefore, the linguistic formulation is

IF *CASE1* THEN *EFFECT1*,

where *CASE1* reflects the input membership variable and its value, and *EFFECT1* reflects the output membership variable and its value. *CASE1* can consist of more than one system input.

The logical operations *AND, OR, NOT* are used to represent more complex relations as shown in Figure 2.11.

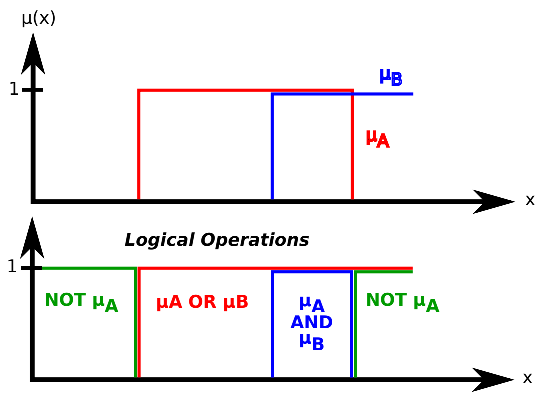


Figure 2.11: Logical operations *AND, OR, NOT* of membership functions.

Therefore, it is possible to combine membership functions, like

IF ***CASE1*** **AND/OR** ***CASE2*** **THEN** ***EFFECT1***,

where *CASE1* could for instance be the *UAV*'s height and *CASE2* the *UAV*s speed.

Next, all defined rules are evaluated. This means that we determine how the truth values of the input cases influence the truth values of the output effects. In our hovering example, the rules can be defined as

1. **IF** *HEIGHT* is FALLING **THEN** *THRUST* is BIGGER
2. **IF** *HEIGHT* is RISING **THEN** *THRUST* is LESS
3. **IF** *HEIGHT* is HOVERING **THEN** *THRUST* is NEUTRAL

The fuzzification delivers the truth values of the input membership functions and during the inference these values are evaluated rule-by-rule to get the truth value of the output membership functions, as illustrated in Figure 2.12.

Defuzzification. The last step of the fuzzy logic control is the inverse process of fuzzification. Here, the output membership function is mapped to a real value for the system input. Several methods exits:

- **Center-of-Area (*CoA*)** : Calculates the center-of-gravity on the area of the composited output function and the centroid mapped to the x-axis is chosen as system input.

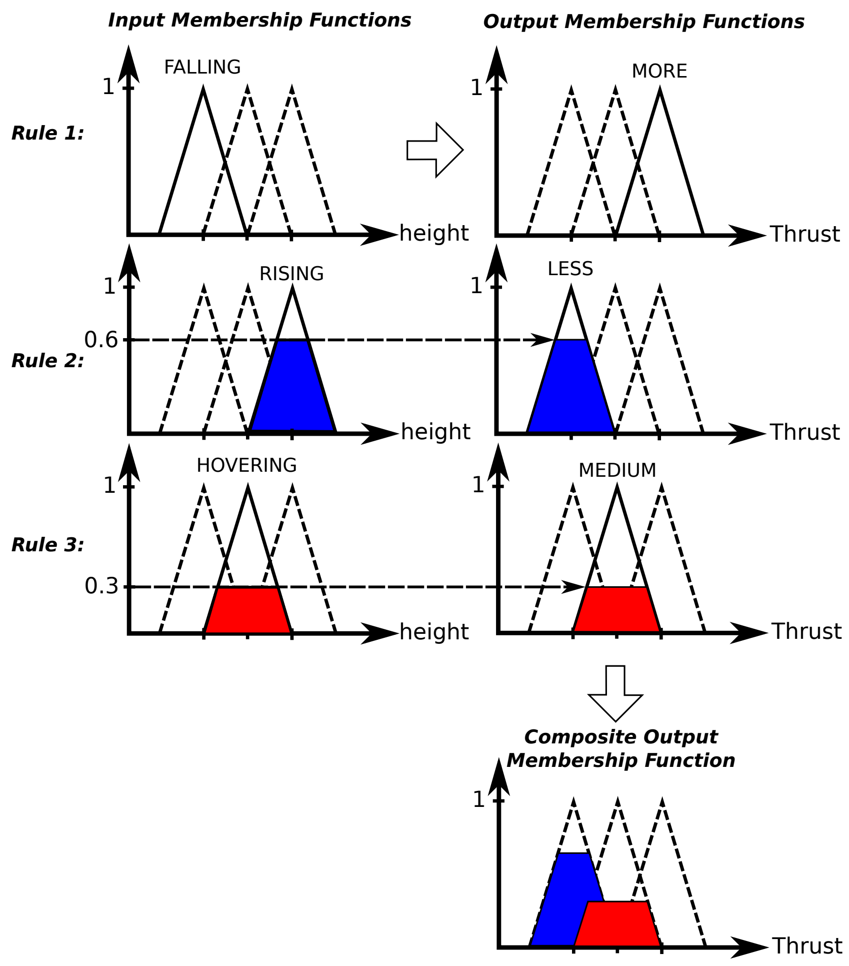

Input Membership Functions
Output Membership Functions
Rule 1:
FALLING
MORE
Rule 2:
RISING
LESS
0.6
Rule 3:
HOVERING
MEDIUM
0.3
1
height
Thrust
Composite Output Membership Function

Figure 2.12: Inference step of fuzzy logic control where the truth values of the input membership are evaluated on the basis of the rule set, and the truth value of the output is estimated for every rule. In the end all outputs are composited to one single output membership function.

- **Center-of-Maximum (*CoM*)** : The maximum of a membership function is mapped to the x-axis, and this value is used as system input.
- **Mean-of-Maximum (*MoM*)** : Calculates the mean of all maxima of the membership functions, projects it to the x-axis, and this value is used as system input.

Figure 2.13 shows the different methods described above. On the left-hand side the *CoA*-, on the middle the *CoM*- and on the right-hand side the *MoM* method is shown, respectively.

The advantage of a fuzzy logic controller is that it is able to control a system in real time while nonlinear as well as only partially known systems can also be handled. It is easy to maintain the

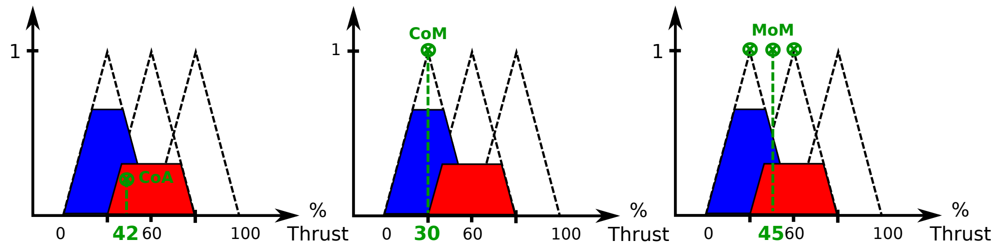


Figure 2.13: Different defuzzification methods: On the left-hand side the different *CoA*, in the middle the *CoM*, and on the right-hand side the *MoM*-method is shown, respectively. The projected points are used as system input to control the system.

controller even it additional information of the process added. Rules can be added or adapted easily. The disadvantage of the fuzzy logic controller is the fine adjustment which can only be done experimentally, and there is no common technique to tune precision. As in every control system, a trade-off between speed and precision has to be found. Due to the advantages of fuzzy logic controllers mentioned above and the fact that the controller design does not depend on the specific type of *UAV*, this type of controller was chosen for the visual servoing system in this work.

2.4.3 Summary

In this chapter, a detailed overview over different approaches for autonomous navigation of mobile robots have been presented. In detail, we discussed visual servoing systems and presented a comprehensive literature review. Furthermore, visual pose estimation and an introduction to control theory were given. In the next chapter, the hardware and software framework of the proposed system is discussed.

Chapter 3

System Components

Contents

Before explaining the implementation in detail, we will give an overview of the available hardware and used software. As an introduction the quadrotor helicopter is described. Thereupon, the characteristics of the used *MAV* for our human-inspired visual servoing approach are given. Additionally, we describe the interface between the *MAV* and the ground-station.

3.1 Hardware

The most important hardware component is the aerial vehicle. In this work, an *MAV* assembled by the company Ascending Technologies GmbH (*AscTec*) [1] is used. The use of such an *MAV* has several advantages for the *PEGASUS* project. With a quadrotor helicopter it is possible to move in arbitrary directions with a very low speed or to hold the current position. In the following, the hardware components of the aircraft are described in more detail.

3.1.1 Quadrotor Helicopter Characteristics

As the name suggests, quadrotor helicopters rely on four rotors that generate the thrust to lift the aircraft. This type of aerial vehicle is classified as rotorcraft in contrast to fixed-wing aircrafts.

[1] http://www.asctec.de

Quadrotor helicopters are vertical take-off and landing (*VTOL*) aerial vehicles like a normal helicopter, but with the benefit of fixed-pitch blades. Conventional helicopters change the pitch angle of the blades mechanically which is quite complex and thus expensive. The quadrotor helicopter has a simpler mechanical construction and the direction can be changed by setting appropriate rotor speeds.

The orientation of the quadrotor helicopter is dynamically defined by three parameters: Roll, pitch and yaw angle as depicted in Figure 3.1.

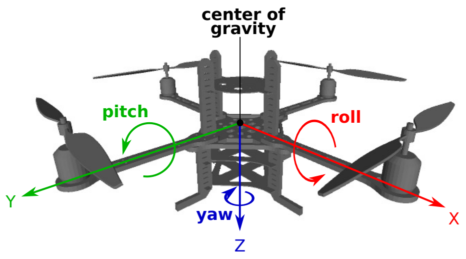


Figure 3.1: Orientation of a quadrotor helicopter body-frame with the angles roll, pitch and yaw.

With the Center-of-Gravity (*CoG*) as origin of the body-frame of the aircraft, the x-axis is defined in flight direction, z-axis in direction of gravity and y-axis follows from the right-handed system. The rotating direction of every two rotor pairs is complementary, which means that one pair rotates clockwise and the other pair anti-clockwise as depicted in Figure 3.2. This gives the aircraft the possibility to hover in the air, if it can be ensured by proper control that the torque θ_i of the two pairs is equal, so

$$\theta_1 + \theta_3 = \theta_2 + \theta_4. \tag{3.1}$$

Assuming identically performing rotors, this requirement results in the necessity of equal rotational speeds ω_i.
The quadrotor helicopter can be moved by changing the dynamic parameters roll, pitch, yaw and thrust as described in Table 3.1. For example, flying in positive x-direction needs a negative pitch angle whereas roll and pitch are zero. Therefore, the rotation speeds ω_2 and ω_4 are held constant, ω_1 needs to be decreased and ω_3 needs to be increased.

3.1.2 AscTec Pelican

Besides the working principle which is common to all quadrotor helicopters, the *MAV* used in this work has some specific features which are described in the following. The key feature

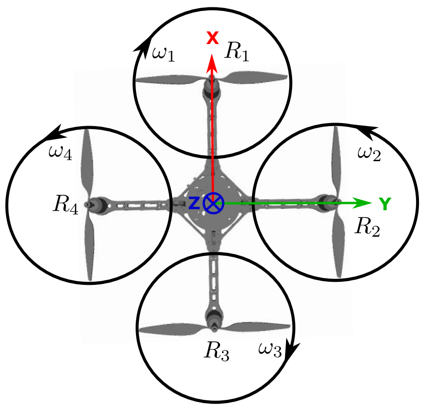


Figure 3.2: Spin direction of the rotors: The rotors R_1 and R_3 rotate clockwise whereas R_2 and R_4 rotate counter-clockwise such that the pair-wise sum of torques is equal as in Equation (3.1).

UAV	roll	pitch	yaw	thrust	ω_1	ω_2	ω_3	ω_4
forward	0	< 0	0	const	$\searrow$	const	$\nearrow$	const
backward	0	> 0	0	const	$\nearrow$	const	$\searrow$	const
right	> 0	0	0	const	const	$\searrow$	const	$\nearrow$
left	< 0	0	0	const	const	$\nearrow$	const	$\searrow$
higher	0	0	0	> 0	$\nearrow$	$\nearrow$	$\nearrow$	$\nearrow$
lower	0	0	0	< 0	$\searrow$	$\searrow$	$\searrow$	$\searrow$
clockwise rotation	0	0	< 0	const	$\searrow$	$\nearrow$	$\searrow$	$\nearrow$
anti-clockwise rotation	0	0	> 0	const	$\nearrow$	$\searrow$	$\nearrow$	$\searrow$

Table 3.1: Relationship between the dynamic parameters roll, pitch, yaw and thrust and the resulting movement direction.

of the *MAV* is its capability to lift a payload of about 500 grams. The flexible modular tower design enables to carry several sensors or extensions, such as laser scanners or, as in our case, an industrial camera. The *MAV* can fly about 20 minutes with one battery charge, where the maximum speed is 14 m/s. It is equipped with two ARM-7 micro processors. The first, called Low Level Processor (*LLP*) is a proprietary module responsible for hardware management and *IMU* sensor data fusion. The second, called High Level Processor (*HLP*) can be used for custom programming. The *LLP* and the *HLP* are connected via a high-speed serial interface for data exchange. The Pelican has several sensors on board, including an *IMU* (accelerometers, gyro sensors), magnetic compass, air pressure sensor and *GPS*.

The *MAV* is additionally equipped with a dual core Intel Atom 1.6 GHz embedded computer, which provides additional computing power for the computationally expensive vision-based algorithms. The embedded computer has 1 GB DDR2 memory and provides interfaces like USB 2.0, a serial interface, a microSD card slot and a mini PCI express Wireless Fidelity (*WiFi*) card based on the 802.11n standard for wireless communication.

The Pelican *MAV* is equipped with the Flight Control Unit (*FCU*) "AscTec Autopilot" that offers three different flight modes:

1. **Standard flight mode**: The Remote Control (*RC*) is used to control the dynamic parameters roll, pitch, yaw and thrust. This mode uses the attitude controller of the *FCU* and the operator is responsible for navigating and for the compensation of position drifts.
2. **Height flight mode**: Autonomously tries to hold the height of the *MAV* based on the fused sensor data. The *RC* can control the angles roll and pitch for movements in a horizontal plane and the yaw angle for the orientation of the *MAV*. The height control is performed by the *FCU*, allowing navigation at a constant height.
3. ***GPS* flight mode**: Can only be used in outdoor environments when a *GPS* signal is available. In this mode, the operator specifies the target position of the *MAV* via the *RC* and the *FCU* directs the *MAV* to the desired position.

For the purpose of vision-based navigation and the actual inspection task, the *MAV* is equipped with an industrial camera (looking forward) as depicted in Figure 3.3, which is stabilized by a pan-tilt unit.

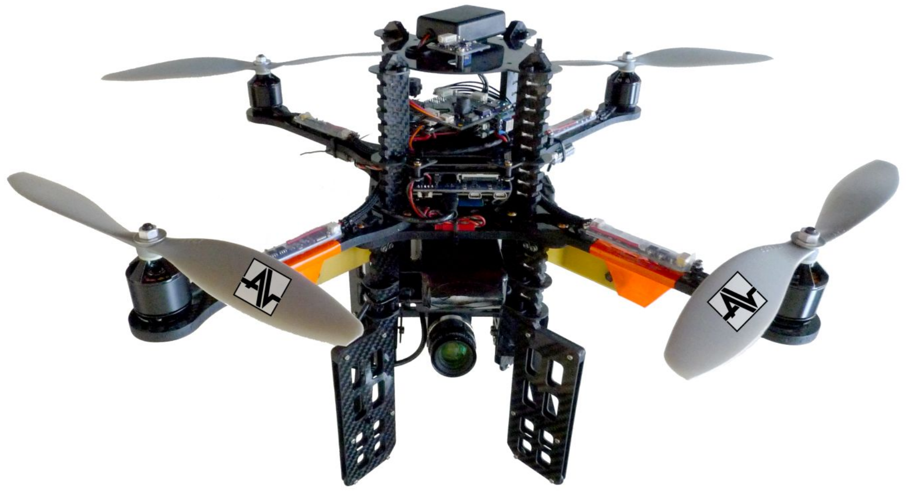


Figure 3.3: Asctec Pelican quadrotor helicopter with a stabilized industrial camera as payload.

The stabilizer ensures that the camera keeps looking in the flight-direction by compensating the pitch and roll independent of the actual flight maneuver. The used camera is an IDS uEye industrial camera with a resolution of 1280×1024 Pixel (*px*). The provided frame rate at full resolution is 25.8 Frames Per Second (*FPS*) and 60 *FPS* when the resolution is reduced to 640×512 *px*. The camera is equipped with a Cinegon 1.4/8 industrial wide angle camera lens.

3.2 Software

The onboard computer of the *MAV* runs Ubuntu Linux 10.04 as basic operating system. On top of the operating system, we use Robot Operating System (*ROS*) [47]. *ROS* is a framework for robotic applications released under BSD license. The framework enables the communication of processes on different systems via message-passing. Thus, a ground-station can send commands to a robot and the robot delivers the measurement data. *ROS* provides a hardware abstraction layer, low-level device control and a variety of commonly used functions. *ROS* can be used to split computational independent tasks and distribute them over several workstations known as *nodes*. The single nodes are connected peer-to-peer, while a single master coordinates communication.

The entire system is depicted by a graph; an example is given in Figure 3.4.

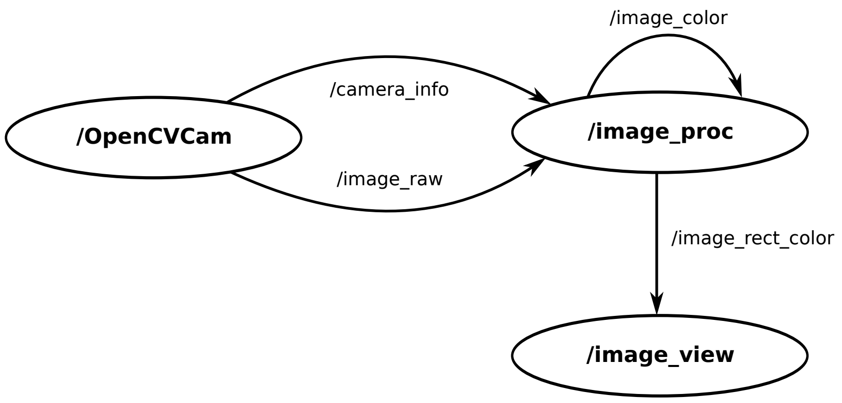


Figure 3.4: A graph-based system overview. The nodes for capturing an image, removing lens distortions and displaying the undistorted images can be distributed over several physical systems. The communication between the nodes is based on topics, whereas some nodes publish data and others subscribe to it.

Three nodes are shown: The *camera node*, the *image processing node* and the *image view node*. Each of them can run on a different system, for instance the robot captures the images, whereas

the ground-station removes lens distortions and displays the resulting image.
With the support of several programming languages and the variety of readily available algorithmic packages, *ROS* is a flexible and powerful platform for mobile robotic applications.

Based on the *ROS* framework we discuss in the next section how our human-inspired visual servoing approach is distributed over different systems. Furthermore, we depict the communication interfaces in detail.

3.3 Communication Interfaces

Before going into detail of our visual servoing approach, we briefly discuss where the several system parts are running. Our system uses a *PBVS* technique for visual servoing as already described in Section 2.2.2. The technique consists of two major parts, namely pose estimation and control. Our system is distributed over multiple devices as shown in Figure 3.5.

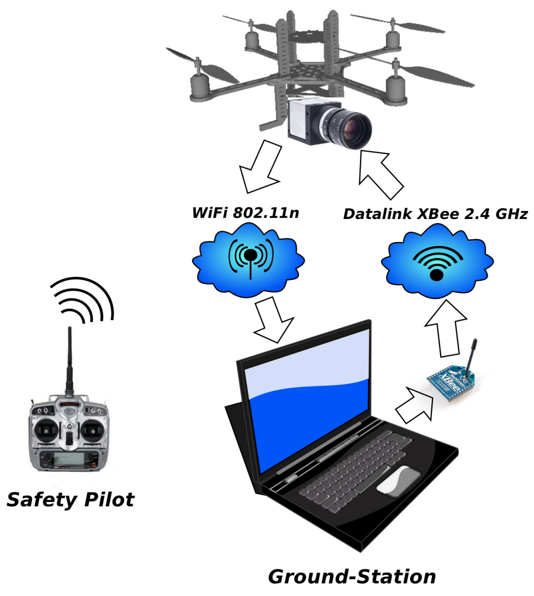


Figure 3.5: Connection interface. The *MAV* captures the images and streams them via *WiFi* to the ground-station. The ground-station computes the current pose based on the images and controls the *MAV* using a wireless datalink. The safety pilot can take over control anytime with the *RC*.

On the *MAV*, the camera captures the images and the onboard computer streams them to the ground-station based on a *WiFi* 802.11n connection. The pose estimation algorithm as well as the *MAV* controller are running on the ground-station, which inturn sends the control commands via a serial wireless connection to the *MAV*. This gives us the advantage that the *MAV* control connection is independent of the traffic resulting from streaming images. The safety pilot can take over control anytime using the *RC*.

The communication of the individual subsystems is based on the *ROS* framework, which has the benefit that each subsystem can be moved individually from the ground-station to the *MAV* and vice versa. However, one has to take care of not overloading the system's computational capabilities.

3.4 Summary

This chapter described the working principle of a quadrotor helicopter and the characteristics of the Pelican helicopter used in this work. Furthermore, the software platform *ROS* which serves as software base for all implementations was briefly described. The next chapter presents the suggested approach for automatic take-off, hovering and landing.

Chapter 4

Human-Inspired Visual Servoing

Contents

This chapter deals with the details of the proposed system and describes the individual blocks. Section 4.1 gives a general overview of the system components. In Section 4.2, the method for position estimation based on the *PTAM* framework, which is a state-of-the-art mono *SLAM* method is described. Moreover, this section discusses the changes and additions to the *PTAM* framework implemented in this work. The fuzzy controller and its implementation are discussed in detail in 4.3. Additionally, the implemented mechanisms for error handling and coping with system failures are presented in Section 4.4. Finally, a typical take-off, hovering, and landing procedure is shown in Section 4.5.

4.1 Overview

The presented system uses a vision-based approach for controlling an *MAV*. The application of computer vision to *UAV*s has gained a lot of interest over the past years and offers several advantages over other sensor modalities. As described in Section 2.2, there are different methods for visual navigation of *MAV*s. This work implements a *PBVS* approach, which requires a mecha-

nism for estimating the current pose of the *MAV*. The reason for choosing a *PBVS* based approach is that these are able to navigate within an unknown environment. In contrast, *IBVS* approaches are not able to navigate within an unknown environment, since they require target frames for navigation.
The proposed modular system design decouples the pose estimation from the controller block which allows for an individual exchange of each block for possible future extensions. Moreover, the encapsulated blocks are easier to handle in terms of maintenance and error analysis. A block diagram of the overall system is shown in Figure 4.1.

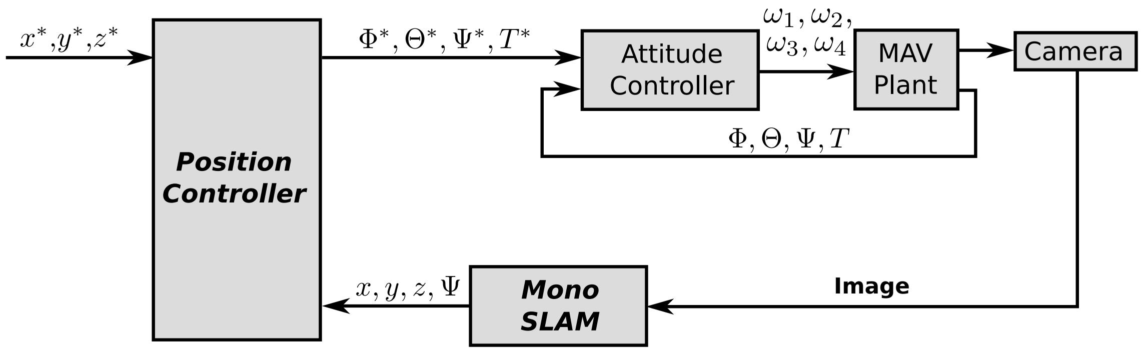


Figure 4.1: System overview: The system represents a cascaded control loop consisting of an Attitude Controller (provided by the *FCU*) and the actual Position Controller. The Attitude Control directly affects the rotation speed of the rotors ω_i, whereas the Position Controller affects the pose of the *MAV* in terms of the angles roll Φ^*, pitch Θ^* and yaw Ψ^* as well as the overall thrust T^*. The input to the control loop is the desired position x^*, y^*, z^*. A camera, mounted on the *MAV* is used to capture images which are then processed by a Mono *SLAM* algorithm to estimate the current pose.

The used *MAV* has an internal attitude controller which directly controls the rotation speed ω_i of the four rotors. This controller ensures that the *MAV* maintains a stable behavior regarding the given angles roll Φ^*, pitch Θ^* and yaw Ψ^*. The attitude controller is fed by the actual position controller with the desired angles as well as the desired system thrust T^*. Hence, the overall system represents a cascaded control loop. The camera mounted on the *MAV* is used to capture images which are then processed by the mono *SLAM* block to estimate the current pose of the *MAV*. The pose estimation therefore works based on the "Inside-Out" principle and estimates the current position and orientation of the *MAV*. These estimates are fed into the position controller to compute the updated actuating variables based on the desired position x^*, y^*, z^*.

After this general system overview, the pose estimation and the position controller as the two main system components are described in detail in the following sections.

4.2 Visual Pose Estimation

The suggested *PBVS* approach to visual servoing requires an estimate of the current pose (position and orientation) of the *MAV*. There are several possibilities to obtain a pose estimate using one or more cameras. These methods can be categorized in model-based and non-model-based systems, as well as hybrid systems. Marker-based approaches are a sub-class of model-based approaches, which rely on the detection of known markers in the scene. These systems suffer from certain limitations, because pose estimation can only be performed when the known markers are visible and detectable in the scene. In contrast, *VSLAM* systems create a map of the environment and perform a localization within that environment on-line. While this localization is still based on a model, it uses natural rather than artificial features. This implicitly allows for an autonomous navigation in unknown environments, since the created map can be extended without prior knowledge about the environment.

The *VSLAM* technique employed in this work uses a single camera for the pose estimation task. Compared to stereo camera sets, a mono camera complies better with the limited payload on *MAV*s. Moreover, stereo cameras do not provide more accurate results due to the short camera baseline compared to the distances in the scene, as discussed in [26]. In addition to the pose estimation and navigation task, the camera is also used for the inspection of power lines and power pylons in the *PEGASUS* project.

Based on the images acquired by the camera, we use the Parallel Tracking and Mapping (*PTAM*) [25] framework by Klein and Murray as state-of-the-art mono *SLAM* approach. *PTAM* is capable of estimating the current camera pose up to the scale. Since the visual navigation of an *MAV* also requires the correct scale, we replaced the initialization process of the *PTAM* framework. The working principle of the standard *PTAM* approach is described in the following. Subsequently, our addition to the *PTAM* framework which allows the determination of the scale factor is described.

4.2.1 Parallel Tracking and Mapping

The fundamental idea of the *PTAM* framework is to make use of the parallel processing capabilities of modern computer hardware by splitting up the localization and mapping task into two parallel working threads. This results in several advantages over previously suggested approaches. First, it provides the possibility that tracking and mapping can be performed in different intervals. Generating an environment map does not have such a high priority due to the limited movement speed, whereas it is vital to have tracking information available in real-time. Second, the mapping task can be skipped during periods where the camera is stationary or in already known regions of the environment. In contrast to other systems [35, 38], this has the advantage that no redundant data filtering needs to be performed. Another property of the *PTAM* framework is that it does not utilize any *EKF* state estimation and hence does not model measurement uncertainties, which reduces computational complexity. These characteristics make the *PTAM* framework well suited for real-time applications.

The map representation within *PTAM* consists of M environment feature points in the world coordinate frame $\mathcal{W}$, stored in homogeneous coordinates, i.e.

$$\boldsymbol{p}_j^{\mathcal{W}} = [x_j^{\mathcal{W}},\ y_j^{\mathcal{W}},\ z_j^{\mathcal{W}},\ 1]^T, \quad j \in \{1 \ldots M\}, \tag{4.1}$$

and N so called key-frames. Key-frames are environment snapshots at a given point t_i in time, comprising

- the camera pose $\boldsymbol{E}_{\mathcal{C}i}^{\mathcal{W}}$,
- a four level gray-scale image pyramid, and
- the list of visible feature points.

The camera pose $\boldsymbol{E}_{\mathcal{C}}^{\mathcal{W}}$ is estimated within the map and is robust to scale changes and partial occlusions. Subsequently, the processes of tracking and mapping are explained in more detail.

Tracking. For every acquired image, *PTAM* performs several steps to estimate the current camera pose. The first step is to compute a four level gray-scale image pyramid which is then used for feature extraction, as shown in Figure 4.2(a). The extracted features points – *FAST* (Features from Accelerated Segment Test) corners [48] – on level 0, 1, 2 and 3 are depicted in red, yellow, green and blue respectively. The reason why FAST corners are used is that they allow fast processing to enable real-time capabilities, while still hovering a reasonable repeatability. *PTAM* then uses a simple motion model to compute a prior estimate of the camera pose. Using this pose estimate and the camera calibration matrix, previously acquired map feature points are back-projected into the image. The next step is to compute the distance between the back-projected feature points and the corresponding image features. Based on the resulting distance measures, the camera pose is updated iteratively with an M-Estimator [49]. The pose update and back-projection steps are performed in a two stage coarse-to-fine manner. The coarse update uses few feature points on lower resolution images, whereas the fine step improves the estimate by using many feature points on a higher resolution level.

This implies that the *PTAM* framework provides a pose estimate $\boldsymbol{E}_{Ci}^{\mathcal{W}}$ for every frame $\mathcal{F}_i$, which allows continuous tracking of the camera pose over time. The pose estimate consists of the current coordinates $[x, y, z]$ as well as the orientation $[\Phi, \Theta, \Psi]$ in a 4×4 matrix representation. The provided matrix is a member of the Lie group SE(3) [50] which guarantees smooth camera pose tracking.

Mapping. Using a monocular camera makes an initialization step necessary. Before any tracking or mapping can be performed, at least two key-frames are necessary to build an initial map. In the *PTAM* framework, this is achieved by a stereo-initialization which requires the user to translate the camera. Tracking feature points during the translation allows to compute the baseline using a homography estimation. The standard initialization assumes a baseline length of $l = 10$ cm and has the disadvantage that the scale factor is unknown. The result of the initialization is a map consisting of two key-frames which can then be extended by new key-frames as the camera moves in the scene. Extension of the map is performed by adding new key-frames, which is subjected to the following three criteria:

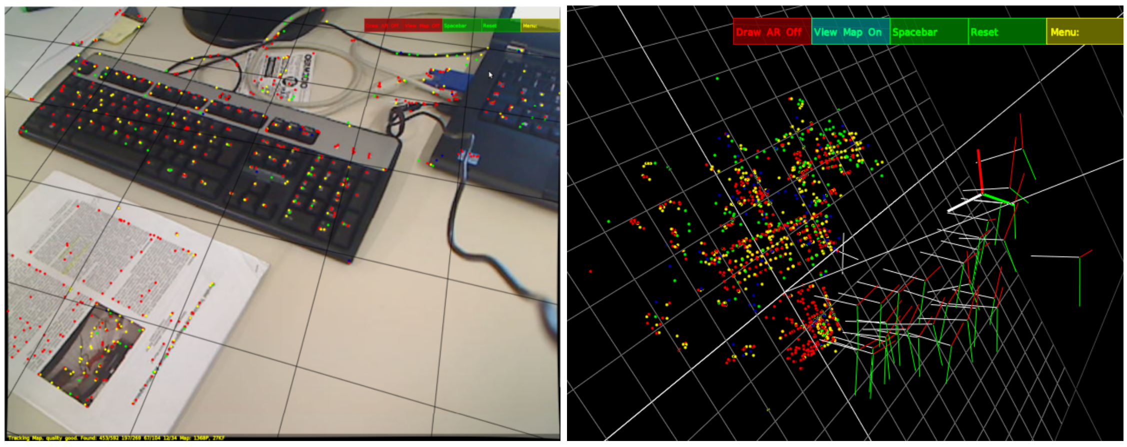
- Good tracking quality: The quality of the pose estimation is assessed for every new frame based on the number of successful feature observations. If the tracking quality is too low, the current frame disqualifies as new key-frame.
- Temporal distance between key-frames: A new key-frame can only be obtained every 20 frames.
- Spatial distance between key-frames: A new key-frame can only be obtained if the spatial distance to every captured key-frame is above a certain threshold.

The last requirement implies that no new key-frames are added in case of a stationary camera pose, which avoids a common *SLAM* problem of corrupt maps if the camera position does not change. The spatial distance threshold depends on the length of the baseline during the initialization process. A longer baseline provides a higher accuracy regarding the triangulation of feature points and hence yields a more accurate map, however it typically results in a map with less initial feature points.
As soon as key-frames are inserted that contain feature points which are not yet triangulated in the map, a triangulation is performed to compute the world-coordinates of these features. The

triangulation is based on correspondences which are established by an epipolar search of the feature points in the associated neighboring key-frames. By inserting the newly triangulated feature points, the map is extended as the camera moves in the scene.

The *PTAM* framework uses a two-step map optimization technique, the so called global and local bundle-adjustment. With every newly added key-frame, a local bundle-adjustment is performed to refine the feature points in the map as well as the camera pose estimates for the k neighboring key-frames. During time periods where no new key-frames are added, global bundle-adjustment is performed to refine all map feature points and the camera pose estimates of all key-frames. Depending on the size of the map, the global bundle-adjustment requires a lot of computational time, since its complexity follows $O(N^2M)$ [25]. However, due to the parallel processing of the tracking and mapping tasks, this does not affect the real-time capability.

An example of a scene with the corresponding map is shown in Figure 4.2. Figure 4.2(a) shows the scene with the extracted feature points, whereas the corresponding map is shown in Figure 4.2(b). The map comprises the estimated camera poses for the key-frames and the triangulated feature points.



(a) PTAM image frame with feature points. (b) Corresponding PTAM environment map.

Figure 4.2: Exemplary scene from the standard *PTAM* framework. The left-hand side (a) shows a scene with extracted feature points. Figure (b) depicts the corresponding map and shows the estimated camera positions for some key-frames.

4.2.2 Scale Factor Determination

In the standard implementation, the *PTAM* approach only yields the camera pose estimate up to a scale factor. The reason for this issue is sketched in Figure 4.3.

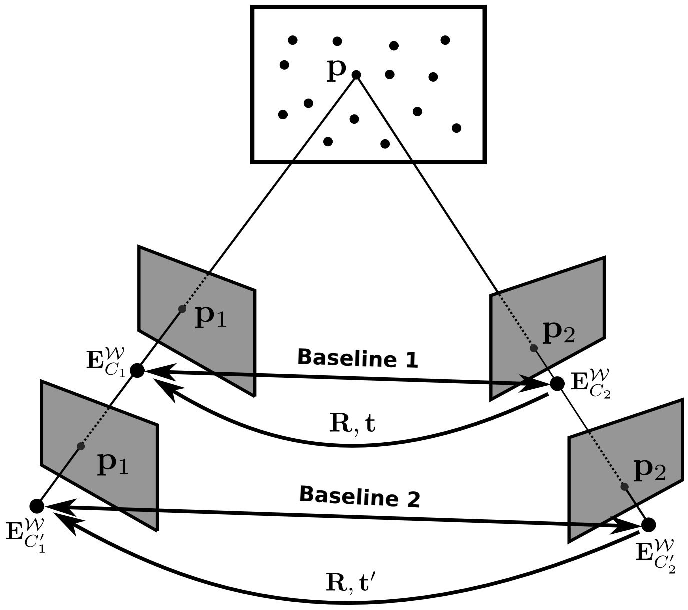

$\mathbf{p}$
$\mathbf{p}_1$
$\mathbf{p}_2$
$\mathbf{E}^{\mathcal{W}}_{C_1}$
$\mathbf{E}^{\mathcal{W}}_{C_2}$
Baseline 1
$\mathbf{R}, \mathbf{t}$
$\mathbf{p}_1$
$\mathbf{p}_2$
Baseline 2
$\mathbf{E}^{\mathcal{W}}_{C'_1}$
$\mathbf{E}^{\mathcal{W}}_{C'_2}$
$\mathbf{R}, \mathbf{t}'$

Figure 4.3: PTAM initialization: *PTAM* estimates a 3×3 homography matrix between the key-frames taken during initialization. Since this homography is not unique with respect to the translation $\boldsymbol{t}$ respectively $\boldsymbol{t}'$ along the baseline and the length of the baseline is unknown, *PTAM* assumes a fixed baseline length and scales the environment accordingly. This results in an unknown scale factor.

During initialization, *PTAM* uses tracked point correspondences to estimate a 3×3 homography $\boldsymbol{H}$ between the first two key-frames. The resulting homography is related to a rotation $\boldsymbol{R}$ and a translation $\boldsymbol{t}$ along the baseline,

$$\boldsymbol{H} = \boldsymbol{R} + \boldsymbol{t}\boldsymbol{n}, \tag{4.2}$$

where $\boldsymbol{n}$ represents the directional vector between the two camera positions $\boldsymbol{E}^{\mathcal{W}}_{C_1}$ and $\boldsymbol{E}^{\mathcal{W}}_{C_2}$. However, the homography between the two key-frames is not unique with respect to the translation $\boldsymbol{t}$. While the rotation matrix $\boldsymbol{R}$ between $\boldsymbol{E}^{\mathcal{W}}_{C_1}, \boldsymbol{E}^{\mathcal{W}}_{C_2}$ and $\boldsymbol{E}^{\mathcal{W}}_{C'_1}, \boldsymbol{E}^{\mathcal{W}}_{C'_2}$ remains the same, the translation vectors $\boldsymbol{t}$ and $\boldsymbol{t}'$ (the baselines) differ. Since the total depth in the scene and the length of the baseline are unknown, *PTAM* assumes a fixed baseline length and scales the environment accordingly.

For navigating an *MAV* in an unknown environment, the scale factor is of vital importance. In order to determine the scale, we modify the initialization process by adding a-priori information in form of a known target (marker) to the scene. This means that during initialization the system relies on a model-based pose estimation technique. However, the model is only required during the acquisition of the first two key-frames. Afterwards, the target can be removed from the scene, and pose estimation solely relies on extracted natural features. The modified initialization utilizes the ARToolKitPlus [32] framework to obtain the metrical camera pose from the first two key-frames during initialization. This process is depicted in Figure 4.4. Since the dimensions of the target are known, the camera pose including the scale factor are estimated.

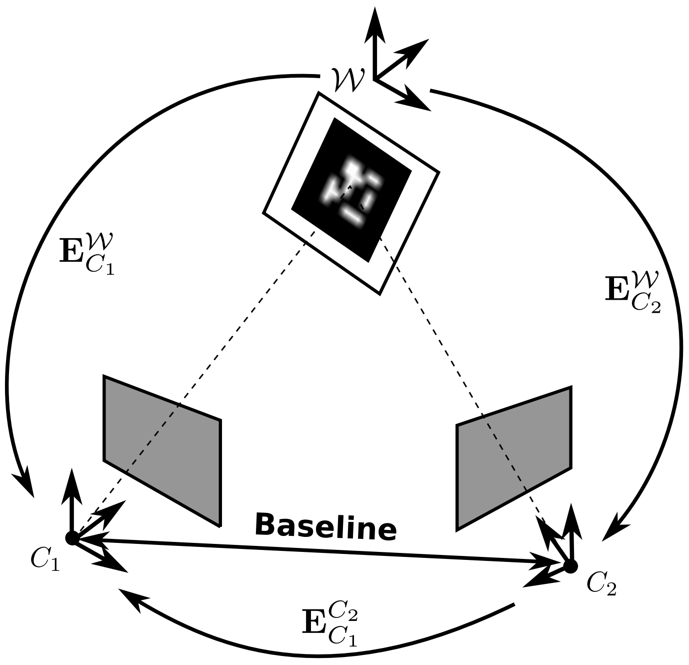

$\mathcal{W}$
$\mathbf{E}_{C_1}^{\mathcal{W}}$
$\mathbf{E}_{C_2}^{\mathcal{W}}$
Baseline
C_1
C_2
$\mathbf{E}_{C_1}^{C_2}$

Figure 4.4: Given the two key-frames from initialization, the camera pose in the world coordinate frame $\mathcal{W}$ can be estimated using the ARToolKitPlus [32] framework. This allows the determination of the metric baseline between the two camera positions.

Given the camera pose $\boldsymbol{E}_{C_1}^{\mathcal{W}}$ estimated from the first key-frame and the pose $\boldsymbol{E}_{C_2}^{\mathcal{W}}$ estimated from the second key-frame, the rigid transformation $\boldsymbol{E}_{C_1}^{C_2}$ can be computed as

$$\boldsymbol{E}_{C_1}^{C_2} = \left(\boldsymbol{E}_{C_2}^{\mathcal{W}}\right)^{-1} \boldsymbol{E}_{C_1}^{\mathcal{W}}. \tag{4.3}$$

With

$$\left(\boldsymbol{E}_{C_2}^{\mathcal{W}}\right)^{-1} = \boldsymbol{E}_{\mathcal{W}}^{C_2}, \tag{4.4}$$

Equation (4.3) becomes

$$\boldsymbol{E}_{C_1}^{C_2} = \boldsymbol{E}_{\mathcal{W}}^{C_2}\boldsymbol{E}_{C_1}^{\mathcal{W}} = [\boldsymbol{R} \mid \boldsymbol{t}] \,. \tag{4.5}$$

Given the known translation $\boldsymbol{t}$ between the two camera positions, *PTAM* yields correctly scaled camera pose estimates and a correctly scaled map. With the correct scale it is now possible to localize the *MAV* in unknown environments based on visual input. However, for navigation not only the pose of the *MAV* is necessary, but a controller is needed. In the next section the control design is discussed.

4.3 Control

The controller design needs to take the equipment used and the characteristics of the target *MAV* into account. As discussed in Chapter 2, there is a lot of ongoing research regarding the control of *MAV*s. Most controller approaches require a mathematical model describing the system dynamics. The quadrotor helicopter used in this work has a built-in, proprietary attitude controller which makes it even more difficult to find a mathematical description of the system. For this reason, the suggested approach comprises a controller that does not need a mathematical system description. We use a human inspired fuzzy logic control, since this offers several advantages. First, a fuzzy controller does not require a detailed mathematical model of the *MAV*. Second, the controller design is not limited to a special *MAV* type and can be adapted easily if the system configuration changes. Third, in contrast to traditional approaches with *PID*-controllers where the proportional, differential and integral factors are often difficult to estimate without a mathematical model of the plant, the design of a fuzzy logic controller is very intuitive. As a result, the rule-set obtained during the design is easily maintainable because rules can easily be added, changed or removed. Finally, fuzzy logic controllers can deal with nonlinear and unstable systems like *MAV*s and are very fast because the implementation is based on a simple look–up–table.

In the following, the initialization of the system and the design of the position controller is discussed, and the controller blocks are described in detail.

4.3.1 Coordinate Frame Initialization

Given the initialization of the pose estimation algorithm as described previously, the coordinate frame of the artificial marker defines the origin for the movement of the *MAV*. The origin can be redefined for *MAV* navigation in order to be independent from the marker position in the environment. This means that after finishing *PTAM*'s initialization, the operator can move the

MAV to the desired position Q_{init} and can define this pose as the new origin for navigation. Since the modified *PTAM* framework already delivers the initial camera pose estimate C_{init} in the world coordinate frame $\mathcal{W}$, $\boldsymbol{E}_{C_{init}}^{Q_{init}}$ is still needed to determine Q_{init}. This is depicted in Figure 4.5.

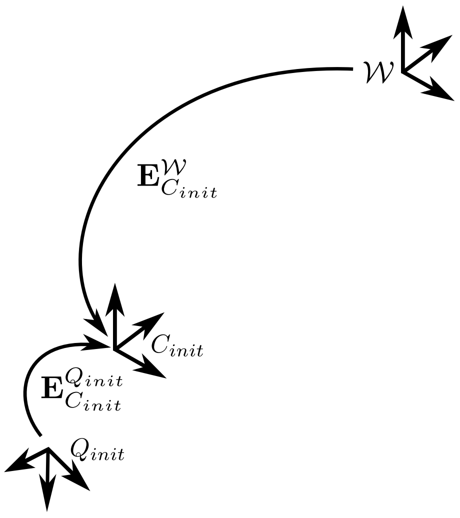

$\mathcal{W}$
$\mathbf{E}_{C_{init}}^{\mathcal{W}}$
C_{init}
$\mathbf{E}_{C_{init}}^{Q_{init}}$
Q_{init}

Figure 4.5: Relationship between the initial *MAV* position Q_{init} and the world coordinate frame $\mathcal{W}$. The operator can define an arbitrary position as coordinate frame origin for the motion controller.

The unknown transformation $\boldsymbol{E}_{C_{init}}^{Q_{init}}$ depends on where and how the camera is mounted on the *MAV* and can be obtained by a calibration step. Since we are using a pan–tilt unit, the attitude is corrected in a way that the camera is always parallel to the ground. Therefore, we are able to directly use the camera position as Q_{init} with the additional need to rotate the coordinate frame of the camera such that the z-axis of the camera faces into the direction of the *MAV*'s x-axis. Thus, this rigid transformation is constant, meaning

$$\boldsymbol{E}_{C_{init}}^{Q_{init}} = \boldsymbol{E}_{C_{current}}^{Q_{current}}. \tag{4.6}$$

The resulting pose of the *MAV* can hence be computed as

$$\boldsymbol{E}_{Q_{init}}^{\mathcal{W}} = \boldsymbol{E}_{C_{init}}^{\mathcal{W}} \left(\boldsymbol{E}_{C_{init}}^{Q_{init}}\right)^{-1}. \tag{4.7}$$

When the aircraft changes its position from Q_{init} to $Q_{current}$ after initialization, it is necessary to compute the updated position with respect to the previously defined coordinate origin. The transformation $\boldsymbol{E}_{Q_{current}}^{Q_{init}}$ can be computed based on the current camera pose estimate $\boldsymbol{E}_{C_{current}}^{\mathcal{W}}$

and the rigid transformations obtained in the initialization step as

$$\boldsymbol{E}^{Q_{init}}_{Q_{current}} = \boldsymbol{E}^{Q_{init}}_{C_{init}} \left(\boldsymbol{E}^{\mathcal{W}}_{C_{init}}\right)^{-1} \boldsymbol{E}^{\mathcal{W}}_{C_{current}} \left(\boldsymbol{E}^{Q_{current}}_{C_{current}}\right)^{-1}. \tag{4.8}$$

By inserting equations (4.4) and (4.6), this can be written as

$$\boldsymbol{E}^{Q_{init}}_{Q_{current}} = \boldsymbol{E}^{Q_{init}}_{C_{init}} \boldsymbol{E}^{C_{init}}_{\mathcal{W}} \boldsymbol{E}^{\mathcal{W}}_{C_{current}} \boldsymbol{E}^{C_{init}}_{Q_{init}}. \tag{4.9}$$

The relations between the corresponding coordinates frames after the aircraft has moved are shown in Figure 4.6.

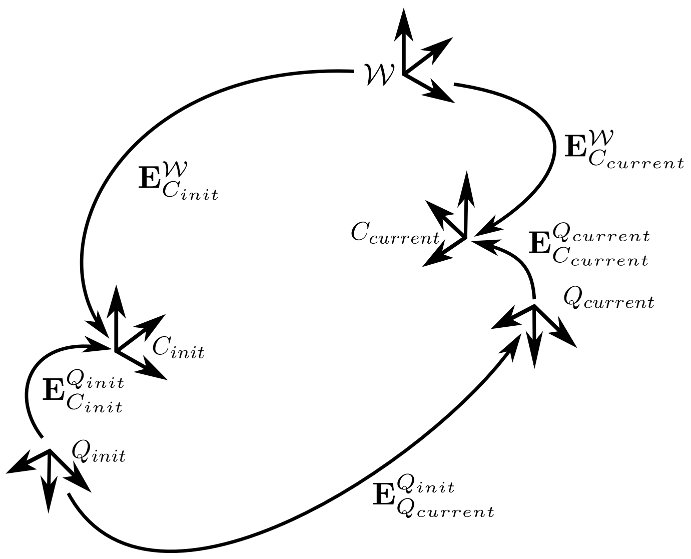

$\mathcal{W}$
$\mathbf{E}^{\mathcal{W}}_{C_{current}}$
$\mathbf{E}^{\mathcal{W}}_{C_{init}}$
$C_{current}$
$\mathbf{E}^{Q_{current}}_{C_{current}}$
$Q_{current}$
C_{init}
$\mathbf{E}^{Q_{init}}_{C_{init}}$
Q_{init}
$\mathbf{E}^{Q_{init}}_{Q_{current}}$

Figure 4.6: The current position of the *MAV* relative to the defined coordinate origin Q_{init} is estimated. The transformation $\boldsymbol{E}^{Q_{init}}_{Q_{current}}$ can be computed based on the current camera pose estimate and the transformations obtained during the initialization step.

With this coordinate transformation it is possible to estimate the current *MAV* position relative to the initial position Q_{init} by means of the visual pose estimation. The *MAV* can then be moved within this coordinate frame by setting a desired position. The initialization requires user interaction with the advantage that the coordinate origin can be chosen arbitrarily. This allows a more convenient control during flight.

4.3.2 Position Control

The position controller is a so called Multiple Input Multiple Output (*MIMO*)-system. The inputs are the desired position (x^*, y^*, z^*) and the estimated current pose (x, y, z, Ψ) of the *MAV*. The outputs of the controller are the desired angles (Φ^*, Θ^*, Ψ^*) and the thrust T^*. Following from the quadrotor helicopter physics and the chosen body coordinate frame, the x-coordinate is controlled by the pitch angle Θ, the y-coordinate by the roll angle Φ and the z-coordinate by thrust the T, yielding a controller structure shown in Figure 4.7. The x-, y- and z-controller have the same internal structure and are based on a *PID* fuzzy logic controller. Due to the inertia of the aircraft the yaw angle Ψ changes slowly, therefore a simple proportional control can be used to keep the *MAV*'s orientation.

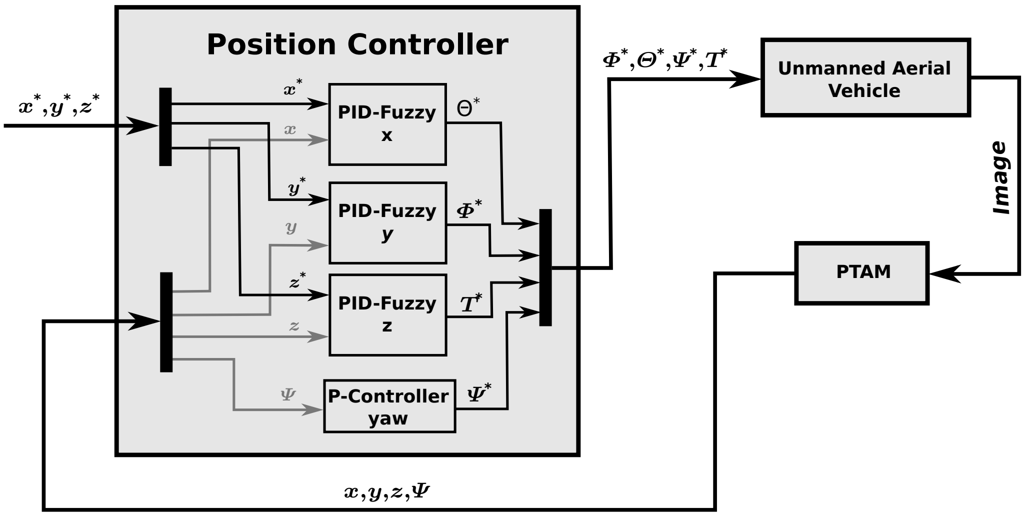


Figure 4.7: The position controller inputs are the desired position (x^*, y^*, z^*), the current position (x, y, z) and the angle yaw Ψ which represents the orientation of the *MAV*. The output of the controller are the desired angles roll Φ^*, pitch Θ^*, yaw Ψ^* and thrust T^*. The position controller contains of independent three *PID* fuzzy logic controllers and a proportional controller to maintain the orientation of the *MAV*.

To be able to control the position in the way we describe it, we use the attitude controller of *FCU* which is a part of the Pelican quadrotor helicopter. The attitude controller selects the rotational speeds of each single rotor such that the desired angles roll Φ^*, pitch Θ^* and yaw Ψ^* are justified. As depicted in Figure 4.7, the position controller consists of four independent controls, namely the x-, y-, z- and yaw- control. However, there is no preference which angle should be controlled first; all are controlled simultaneously.

4.3.3 PID Fuzzy Logic Control

The structure of the sub-block *PID* fuzzy logic control of the position control is shown in Figure 4.8 for the x-coordinate. The controllers for the y- and z-coordinate are implemented identically.

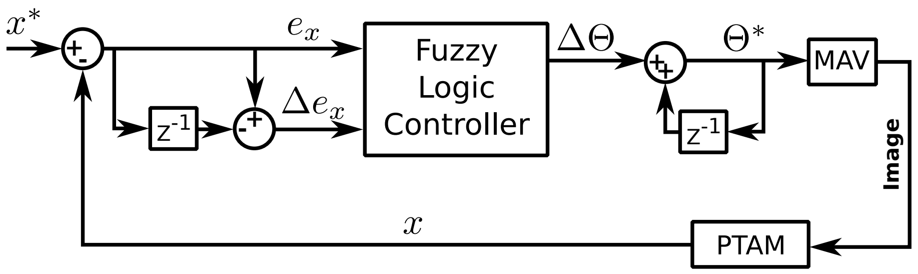


Figure 4.8: PID fuzzy logic block diagram: The x-position error e_x is calculated based on the desired x^*-position and the current x-position from the feedback-control. The fuzzy logic control has two inputs: the position error e_x and the change of the position error Δe_x. The output of the fuzzy logic controller is the change in pitch angle $\Delta\Theta$ which is integrated over time and send to the quadrotor helicopter.

Each controller block is a so called Multiple Input Single Output (*MISO*)-system. Based on the desired position x^* and the current position x, the error e_x and the error change Δe_x can be calculated. Thus, two variables serve as controller input, whereas the output is the desired pitch angle Θ^*. The error change

$$\Delta e_x[n] = e_x[n] - e_x[n-1], \tag{4.10}$$

where $e_x[n]$ is the error at the current time n and $e_x[n-1]$ is the error at the time $n-1$, is defined as the variation of the position error after one time-step. Equation (4.10) is an approximation of the error derivative. The desired pitch angle Θ^* is obtained by integrating the relative change of the pitch angle over time, i.e.

$$\Theta[n]^* = \Theta[n-1]^* + \Delta\Theta[n]. \tag{4.11}$$

The integration part is necessary to compensate *MAV* drifts along the roll angle. Moreover, the temporal integration of the fuzzy logic output enables to cope with external influences such as wind, battery drain or turbulences.

A typical fuzzy logic controller consists of three blocks, namely fuzzification, inference and defuzzification as already introduced in Section 2.4.2.
It has two inputs and a single output variable. First, the input and output variables have to be fuzzificated, meaning that the measured values are mapped to the truth values of the fuzzy set.

The membership functions of the input error e_x consist of the linguistic variables *BIG_NEG, NEG, CENTER, POS* and *BIG_POS* as shown in Figure 4.9.

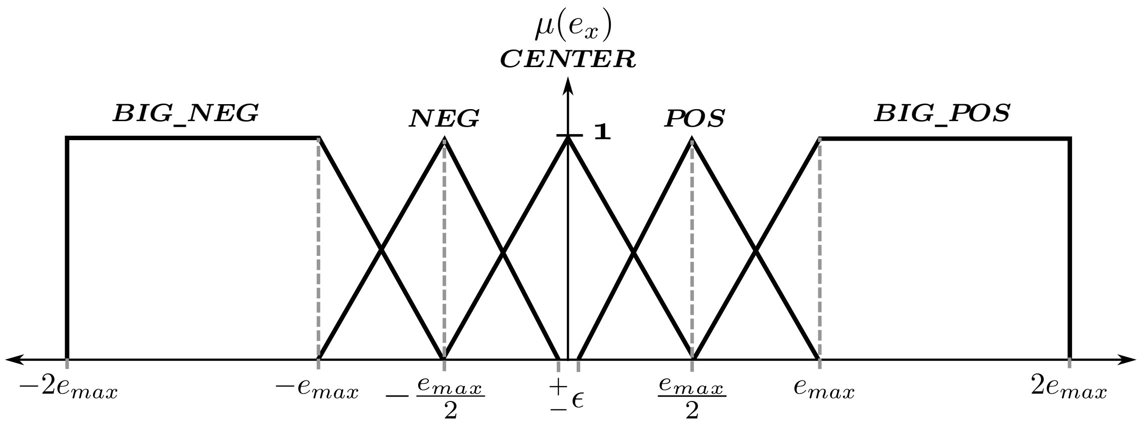


Figure 4.9: Fuzzification of the input error e_x: The *BIG_NEG, NEG, CENTER, POS* and *BIG_POS* are the membership functions to estimate the truth of the fuzzy set in the case of the input error e_x. The width of the membership functions are constrained by e_{max}, which is determined experimentally.

The truth value $\mu(e_x)$ is normalized between zero and one, and the width of the linguistic variables depends on the user-defined maximum value e_{max} of the error. e_{max} is tuned during test flights to get a good flight behavior. The major advantage of fuzzy logic is that the design of the membership functions can be chosen intuitively. Therefore, ϵ can be chosen in a way that it defines the noise level of the pose estimation.

The membership functions for the second input, the error change Δe_x, consist of three linguistic variables, namely *NEG, CENTER* and *POS* as shown in Figure 4.10.

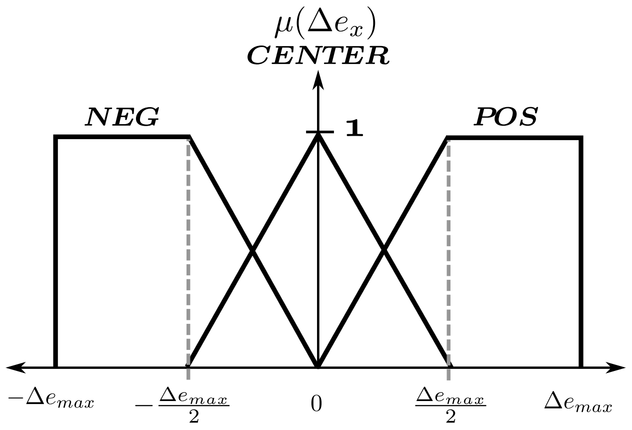


Figure 4.10: Fuzzification of the second input, the error change Δe_x: Three membership functions *NEG, CENTER* and *POS* are used, whereas Δe_{max} is again determined during experiments.

The membership functions are again normalized between zero and one, whereas the width depends on the factor Δe_{max}. This factor is used to get the controller's desired damping. The error change Δe_x is near to the noise level and therefore only three membership functions are used. The *NEG* and *POS* linguistic variables are used to damp the system in case the position error is small but the *MAV* is currently too fast. Thus, the overshoot is kept as low as possible.

As a last step of the fuzzification the linguistic output of the fuzzy controller is designed as depicted in Figure 4.11.

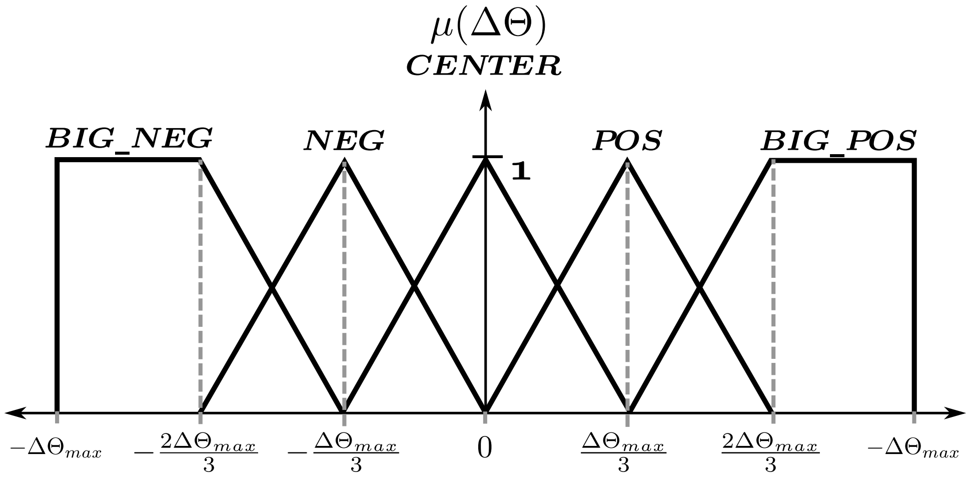

$\mu(\Delta\Theta)$
CENTER
BIG_NEG
NEG
POS
BIG_POS
1
$-\Delta\Theta_{max}$
$-\frac{2\Delta\Theta_{max}}{3}$
$-\frac{\Delta\Theta_{max}}{3}$
0
$\frac{\Delta\Theta_{max}}{3}$
$\frac{2\Delta\Theta_{max}}{3}$
$-\Delta\Theta_{max}$

Figure 4.11: Fuzzification of the incremental pitch output $\Delta\Theta$. $\Delta\Theta$ consists of five membership functions to map the fuzziness to real exact values of $\Delta\Theta$ for the control of the x-position of the *MAV*.

Therefore, five different membership functions are used for the output $\Delta\Theta$. Again, the width of the membership functions is tuned with the factor $\Delta\Theta_{max}$, determined during the test flights.

Once the inputs and the outputs are fuzzificated, the rule-set of the inference block is designed. The truth values are now combined logically with an ***AND*** operator. The entire rule-set is defined in a way an expert would control the pitch Θ of the *MAV* to control the x-position as shown in Table 4.1. There, the membership functions of the input error e_x are defined row-wise and the membership functions of the input Δe_x column-wise. The two input membership functions are combined with a logical **AND** operation to get the appropriate membership function of the output $\Delta\Theta$. Table 4.1 can be read in a linguistic way such as:

IF e_x is *BIG_NEG* **AND** Δe_x is *NEG* **THEN** $\Delta\Theta$ is *BIG_POS*,

in other words if the position error is large and negative, and the velocity error is also negative

$\Delta e_x \backslash e_x$	*BIG_NEG*	*NEG*	*CENTER*	*POS*	*BIG_POS*
NEG	*BIG_POS*	*POS*	*POS*	*CENTER*	*CENTER*
CENTER	*POS*	*POS*	*CENTER*	*NEG*	*NEG*
POS	*CENTER*	*CENTER*	*NEG*	*NEG*	*BIG_NEG*

Table 4.1: The table shows the fuzzy logic rules to control the incremental pitch angle $\Delta\Theta$ of the *MAV*. In the first row and column the two inputs e_x and Δe_x of the controller are defined with their membership functions. The intersection cell of the corresponding membership function of the inputs e_x and Δe_x is the corresponding output membership function $\Delta\Theta$.

then the desired output is large positive. Based on the five membership functions of the first input e_x and three of the second input Δe_x, fifteen rules exist to control the x-position of the *MAV*.

Finally, the last block of the fuzzy logic control is the defuzzification. Therefore, the truth values of the output membership functions are mapped to a real value for the system input. In case of the x-position controller the value of the pitch angle is calculated. We use the *CoA* method as discussed in Section 2.4.2 for defuzzification, where the center-of-gravity of the area of the composited output function is mapped to the x-axis and chosen as system input.

Now it is possible to evaluate the measured values of the inputs (x-position error e_x and error change Δe_x) and calculate the truth of the membership functions. Based on the defined rules the output membership function can be calculated, and fuzzificated to a real value, which is the final output of the fuzzy system. As already mentioned, fuzzy logic control can be used for real-time applications because the whole controller is implemented as a look-up table as depicted in Figure 4.12. The x-axis is the input of the position error e_x and the y-axis represents the input of the change of the position error Δe_x. Based on the surface it is possible to retrieve the fuzzy logic output $\Delta\Theta$ for every combination of the inputs. An arbitrary discretization of the surface can be chosen; however, the smaller the steps are, the more accurate is the resulting controller. Due to the fact that the membership functions are chosen symmetrically, the surface is symmetrical as well.

To conclude, we have discussed all parts of the position controller as well as the *PID*-fuzzy logic control for the x-position control of the *MAV*. The same structure and fuzzy logic rules are used for the y-position and the z-position control, with the inputs being the position error e_i and the change of the position error Δe_i, where i is the index for either x, y or z. Therefore, the angles roll Φ and pitch Θ as well as the thrust T for height control are determined.

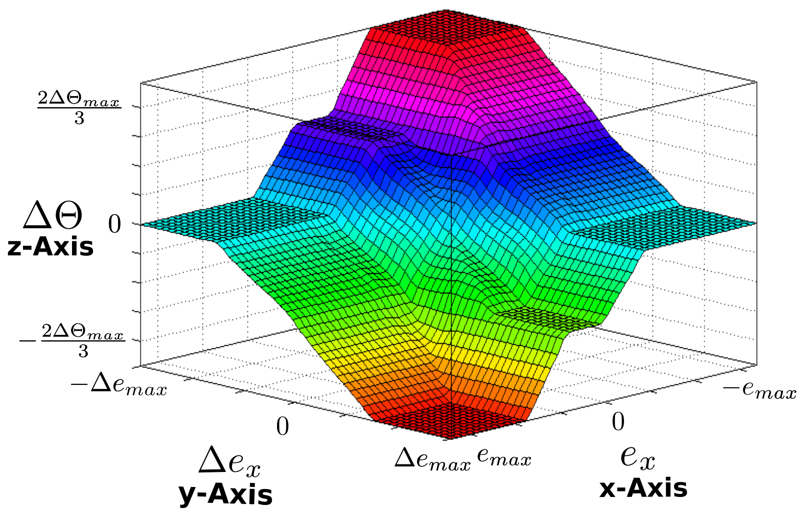


Figure 4.12: Fuzzy logic surface: The x-axis is the input of the position error e_x and y-axis is the input of the change of the position error Δe_x; the z-axis is the fuzzy logic output $\Delta\Theta$ used to control the x-position of the *MAV*.

To fully control the entire position, the yaw angle Ψ of the *MAV* has to be adjusted as well. This controller is described in the next section.

4.3.4 Yaw Control

For the yaw angle Ψ a simple proportional controller is used. The rotation of the *MAV* around the z-axis is very slow due to inertia. Therefore, a proportional controller can be applied without causing a large overshoot. Based on the initial pose's yaw Ψ_{init}, the current yaw angle $\Psi[n]$ is measured. The yaw controller is designed such that the yaw error

$$e_\Psi[n] = \Psi_{init} - \Psi[n] \tag{4.12}$$

is minimized. Thus, we try to keep the initial yaw constant. Theoretically, the initial yaw angle can be set to an arbitrary value any time. In this case the control error $e_\Psi[n]$ is equal to the current yaw $\Psi[n]$ angle and with the proportional gain value K_P the desired yaw $\Psi^*(n)$ angle, which is sent as a control command to the *MAV*, can be calculated as

$$\Psi^*[n] = K_P\, e_\Psi(n). \tag{4.13}$$

By setting $\Psi_{init} = 0$, the yaw controller makes sure that the *MAV* looks towards the known environment and thus a pose can be estimated during take-off and landing procedures.

Before we discuss a typical take-off, hovering, and landing procedure we show next how to deal with system failures.

4.4 Dealing with System Failures

In this section we analyze possible system failures and how to detect and react to these errors. The first step is to detect the errors and then handle them appropriately. Additionally, we have to distinguish between in- and outdoor flights.

4.4.1 Detecting Failures

In the following we discuss how to detect connection errors and pose tracking errors. Furthermore, we picture some possible error scenarios and their influence on our system.

When a connection error occurs, it is impossible to determine if the connection is broken or if the hardware is defective. Both cases have the same effect of no images reaching the ground-station. It no images are received, the pose estimation does not deliver an updated pose of the *MAV*. This error is detected easily, however small delays can appear due to the wireless connection link. If no pose is received within a predefined time interval this is interpreted as connection error. On the other hand, problems can occur if the wireless *MAV* control connection fails, namely the Xbee-datalink. The rate of the control commands received by the *MAV* drops down rapidly. This is automatically detected by the *FCU*.

The pose estimation algorithm can lose track of the environment. This can have different reasons such as the *MAV* making too fast movements or rather big rotations around the z-axis. These movements cause new views of the camera which are not yet in the map. Furthermore, large illumination changes can cause problems, as well as too homogeneous regions resulting in too few features for pose estimation. These errors are easy to detect because our pose estimation framework automatically detects those cases. If this happens repeatedly over a predefine time, a pose tracking error is triggered.

The pose estimation algorithm calculates a pose based on the features extracted from the image. It can happened that similar features are detected over different scales in the image-pyramid, which has the effect of the pose getting invalid. This kind of problem is not as easy to detect as the already discussed failures. However, an invalid pose estimate is very dangerous because the controller gets a completely incorrect input, which in the worst case can cause the *MAV* to crash. Therefore, additional knowledge is necessary to identify such failures.

First, scale changes are monitored using the physical characteristics of the *MAV*, namely the maximum speed. Therefore, the pose is estimated for every frame and labeled with its current time stamp. With this information the speed v_i of the *MAV* between two consecutive frames can be computed and compared to the maximal speed v_{max}. If $v_i > v_{max}$ then the last pose estimate is invalid and the frame is dropped. Then, the next frame is evaluated again. After a predefined time of no valid pose estimations, we trigger a pose tracking error.

Second, for outdoor flights we additionally use the *GPS* signal to verify pose estimation. Thus, during the initialization step, the actual *GPS* position and the yaw Ψ provided by the compass sensor are stored as reference data. The yaw angle Ψ is defined such that it is zero when the *MAV* is aligned to the north pole. Based on the reference *GPS* position a local tangent plane to the Earth's surface is estimated, and the current *MAV* position is calculated with respect to the initialization point. Therefore, we use a local Earth-Centered, Earth-Fixed (*ECEF*) [51] coordinate representation of the *GPS* signal. This representation shows a good accuracy, especially in the case of small position changes compared to the reference position. Based on the reference yaw, the visual pose coordinate system is rotated into the reference *GPS* system where the current pose estimation is checked for validity. Tacking the uncertainty of the reference and the current *GPS* position into account, the visual pose estimation is roughly verified.

4.4.2 Handling Failures

We distinguish between indoor and outdoor failure handling. Outdoors we have the possibility to use additional sensor information such as *GPS* to react to system failures.

If a system failure occurs during an indoor flight, the operator gets an audio-feedback and the *MAV* lands autonomously. During the landing procedure, the fuzzy *PID* integration values are sent to the *MAV* to hold the $x - y$- position. To lose height, the integrated thrust for hovering is slightly reduced. However, if the control connection is broken, the *MAV* control is given back to the *FCU*. Therefore, during all flight experiments we use the *height flight mode*. This mode tries to hold the *MAV* in the current height autonomously, using the air-pressure sensor and *IMU* data. However, these sensors drift significantly over time so in general the safety pilot need to take over control.

If system failures occur during outdoor flights, we use the possibility of autonomous *GPS* navigation. In this case the control is switched to *GPS* position hold, where the internal *FCU* tries to hold the current position of the *MAV*. The operator gets a visual as well as an audio-feedback that an error has occurred and he can take over control using the *RC* to land the *MAV*. This is only possible outdoors where a valid *GPS* signal is available. If no *GPS* signal is available, then the *FCU* automatically switches back to the *height flight mode* and the safety pilot has to take over control in order to land the *MAV*.

During testing and tuning the position controller we used some kind of a fishing rod as shown in Figure 4.13, to ensure that the *MAV* cannot escape. Moreover, this also enables an emergency shutdown, where all rotors are switched off and the *MAV* falls into the rod without damaging it.

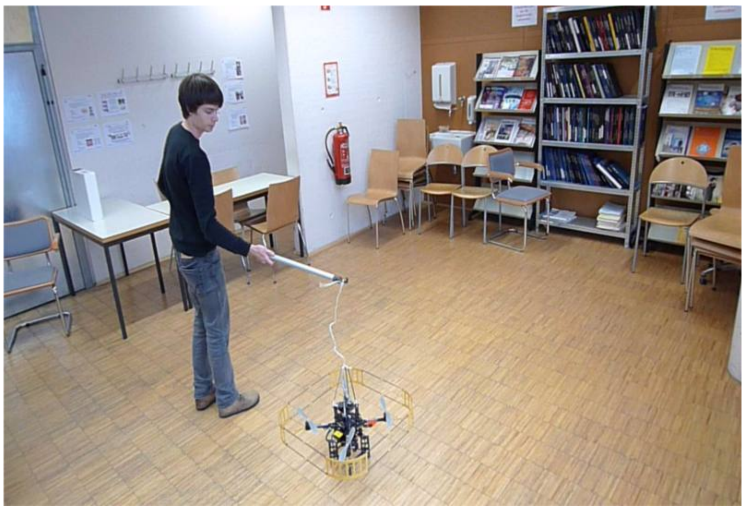


Figure 4.13: Safety feature for testing and tuning the position controller. The fishing rod is a back-up in case of an emergency shutdown.

4.5 A Typical Take-Off, Hovering, and Landing Procedure

After discussing possible system failures and the implemented countermeasures, this section describes how the position controller operates during a sequence of automatic take-off, hovering and landing. As already mentioned, the controller needs an environment map which is created by means of operator interaction. This means that *MAV* camera needs to be translated by the operator in order to build the initial map in the surroundings of the take-off position. The map is then extended during the later flight. After initialization, the new origin for the controller is defined by the operator within the previously built map. The new origin is at the same time the first waypoint after take-off where the *MAV* holds its position. A typical command would be to rise 2 meters and hold position there.

A typical take-off, hovering and landing sequence can be described by a state machine as depicted in Figure 4.14. After the definition of the new origin, the *MAV* is placed on

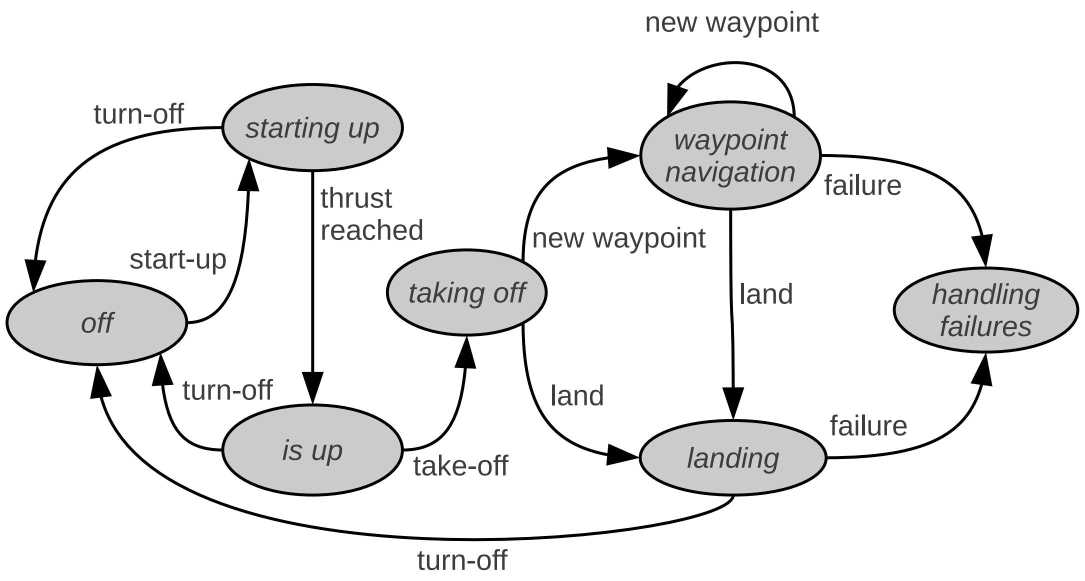


Figure 4.14: Statemachine for automatic take-off, hovering, visual waypoint navigation and landing.

the ground with engines turned off. This corresponds to the state *off* in Figure 4.14. When the engines are turned on, the *MAV* transitions to the state *starting up* and increases the thrust step by step, until the point is reached when the *MAV* almost takes off. The state transition to the *taking off* state is triggered by an operator interaction. Since the camera mounted on the *MAV* is facing forward, there is the possibility that the camera pose cannot be estimated when the *MAV* is placed on ground. In order to overcome this problem, the thrust is further increased such that the *MAV* takes-off and flies to a maximum height of $h = 2\,\text{m}$. If no valid pose estimate is obtained during take-off, the transition from *taking-off* to *landing* is performed. As soon as a valid pose estimate is available, the state transition from *taking-off* to *waypoint navigation* is performed and the *MAV* flies to the first waypoint. If no further waypoints are specified, the *MAV* holds the position at the previously defined waypoint. New waypoints are defined relative to the origin and can be added at any time by the operator, which enables us to fly arbitrary trajectories in the environment as long as the pose can be estimated. The *MAV* can land anywhere in the environment after a particular waypoint-based trajectory flight if the structure of the ground floor permits it. When the landing procedure is triggered, the thrust value is reduced such that the *MAV* slowly reduces height and lands safely on the ground.

When a failure is detected, the *MAV* switches to the *failure handling* state and countermeasures are performed as discussed in detail in the previous chapter.

4.6 Summary

In this chapter we have given an overview of our human-inspired visual servoing approach for automatic take-off, landing and hovering of *MAV*. We discussed the pose estimation algorithm and the control design, which does not need a detailed mathematical model of the *MAV*. Finally, we have discussed how to detect and handle system failures appropriately and as well how to switch between different operation modes based on a state machine representation. In the next chapter, we will present the conducted experiments and give an in-depth discussion of the obtained results.

Chapter 5

Experiments and Results

Contents

Several experiments in a simulation environment as well as indoors and outdoors have been performed. A simulation framework [52] is used to evaluate the robustness of the controller to pose estimation noise and to evaluate how the controller parameters influence the *MAV* dynamics. The precision of hovering and flight trajectories are evaluated in indoor and outdoor environments and compared to other state-of-the-art approaches. The ground-truth for some of the indoor experiments has been established by an off-the-self outside–in tracking system. Evaluation is performed on a ground-station with an Intel Core i5 2.66 GHz processor running Linux.

5.1 Parameter Setup

Table 5.1 shows the parametrization of the position controller for the conducted experiments. As already discussed in the previous chapter, the *PID*-Fuzzy controllers for x, y, and z have the same structure and only differ in the parametrization. The parameters were fitted to the requirements of the simulator, respectively the control interface of the

parameters	Controller		
	PID-Fuzzy x	*PID-Fuzzy* y	*PID-Fuzzy* z
e_{max}	3.4	3.4	1.2
ϵ	0.005	0.005	0.005
Δe_{max}	0.06	0.06	0.03
$\Delta\Theta_{max}$	1300	-	-
$\Delta\Phi max$	-	1300	-
$\Delta T max$	-	-	450
Θ^*_{max}	± 650	-	-
Φ^*_{max}	-	± 650	-
T^*_{max}	-	-	± 600

Table 5.1: Controller parametrization: Input fuzzification parameters and dynamic range of the controller outputs. Each controller output is bound to a given maximum value $(\cdot)^*_{max}$, given a dynamic range of ± 2048.

MAV, regarding the dynamic range of the inputs. The output of each controller block is bound to a maximum output value $(\cdot)^*_{max}$ for safety reasons. The proportional yaw controller, described in Equation (4.13), is parameterized with $K_P = -1000$. The chosen parameters have been found by empirical means during several simulation runs and test flights.

Using this configuration, the performance of the controller was first evaluated in a simulator which has the benefit of a fully controlled environment that allows for a comfortable analysis of the behavior and the sensitivity to certain input factors.

5.2 Simulator

We use a simulation framework [52] to evaluate the robustness of our position controller to pose estimation noise. The simulator configuration is shown in the block diagram of Figure 5.1. The dynamics of the *MAV* are encapsulated in a black–box manner and can be influenced by controlling roll, pitch, yaw and thrust. The simulator yields the current position and orientation of the *MAV*. For evaluating the robustness against pose estimation errors, uniformly distributed noise is added to the position estimate.

Simulation noise. Noise generation for the simulator experiments is based on a two–state random process. This allows us to define the noise level as well as the frequency to closely simulate perceptual noise. While the amplitude follows a uniform distribution $\mathcal{U}(-\widehat{x}, \widehat{x})$, the frequency is determined by a Bernoulli experiment [53]. For each pose, a Bernoulli trial with the outcome P_F

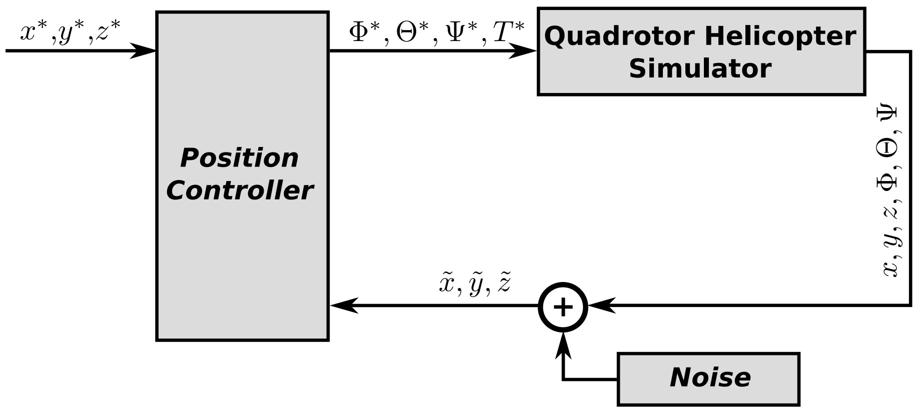

x^*,y^*,z^*
$\Phi^*, \Theta^*, \Psi^*, T^*$
Quadrotor Helicopter Simulator
Position Controller
$x, y, z, \Phi, \Theta, \Psi$
$\tilde{x}, \tilde{y}, \tilde{z}$
+
Noise

Figure 5.1: Simulator overview. The simulator yields the position and orientation of the quadrotor helicopter which is then artificially corrupted by additive noise. The controller influences the dynamics of the simulated *MAV* by setting the roll, pitch and yaw angle as well as the thrust.

is performed. Based on this outcome, it is decided whether a new noise level should be generated. This results in a definition for the noise vector

$$\boldsymbol{u}[n] = \begin{cases} \mathcal{U}(-\widehat{x}, \widehat{x}) & P_F \leqslant p_F \quad \text{or} \quad n = 0 \\ \boldsymbol{u}[n-1] & \text{else.} \end{cases} \tag{5.1}$$

Changing the parameter p_F allows to set the frequency for generating noise levels. For example, a value of $p_F = 0.9$ means that a new noise level is chosen in 90% of all computed positions. The amplitude of the noise can be determined by means of the parameter $\widehat{x}$. The noise is added to the position $\mathbf{x}[n]$ computed by the simulation engine, resulting in

$$\widetilde{\boldsymbol{x}}[n] = \boldsymbol{x}[n] + \boldsymbol{u}[n]. \tag{5.2}$$

The corrupted position $\widetilde{\boldsymbol{x}}[n]$ serves as the feedback input for the position controller.

Based on this setup, the simulator experiments are split into two main parts. First, the performance of the system is evaluated in a hovering scenario. The second experiment demonstrates the flight along a predefined trajectory. Both types of experiments are conducted with and without noise to assess the robustness against pose estimation errors.

5.2.1 Hovering

In the experimental scenario, the *MAV* starts at position $\mathbf{x} = [0\ 0\ 0]^T$ with a target position of $\mathbf{x}^* = [0.5\ 0.5\ 0.5]^T$, where the *MAV* hovers for a period of $\tau = 30\,\text{s}$. As performance metric, the Root Mean Square (*RMS*) error of the absolute position is computed as

$$\textbf{RMS} = \sqrt{\frac{1}{N}\sum_{n\in N}\|\varepsilon_n\|^2}, \tag{5.3}$$

where ε_n denotes the position error vector at time step n. The *RMS* error is computed separately for the height z, the horizontal xy plane as well as in all three dimensions. In order to ignore the transient behavior while the *MAV* approaches the desired position, the *RMS* computation starts in the steady state.

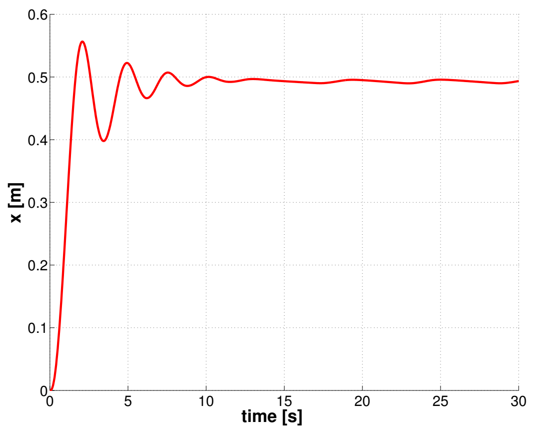
The results for a hovering experiment with noise-free position estimates is shown in Figure 5.2 for the x, y and z coordinate over time. Additionally, Figure 5.2(d) shows the 3D representation of the flight trajectory. Note the almost linear path from the starting position to the desired hovering position, which is possible due to the independent control of x, y and z. The steady state is reached after about 10 seconds. Since the controller for the x and y position operate in the same way and the simulation environment provides ideal conditions, the resulting trajectory for x and y are equal. In contrast to that, the z coordinate shows a different behavior: The design of the thrust controller was done in a way that overshoots are avoided since this is critical during start and especially during landing.

The results regarding *RMS*- and maximum error are summarized in Table 5.2. For this simulation setup, the *RMS* error is bound to about 1 cm with a maximum error of 1.5 cm.

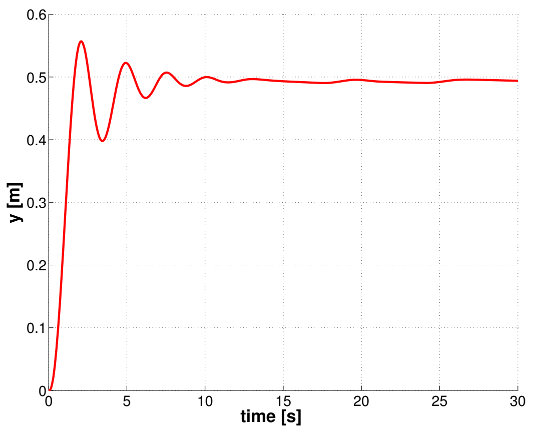
error [m]	z	xy	xyz
RMS	0.0030	0.0098	0.0103
max	0.0033	0.0137	0.0141

Table 5.2: Evaluation of hovering precision without noise. The *RMS*- as well as the maximum error are evaluated for the height z, the horizontal xy-plane as well as in all three dimensions.

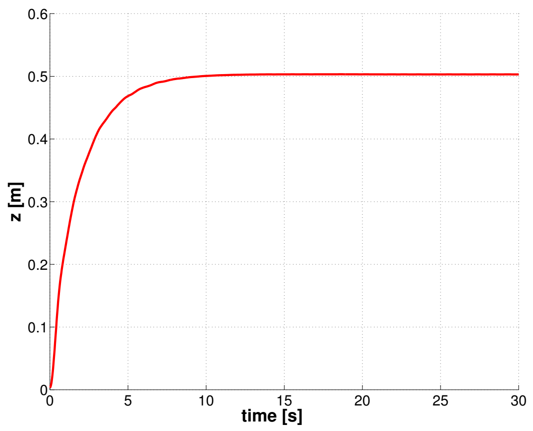
The same type of experiment was performed with additive noise as described above. The noise parameters are chosen as $p_F = 0.4$ and $\widehat{x} = 0.04$ m. This means that there is a 40% chance of



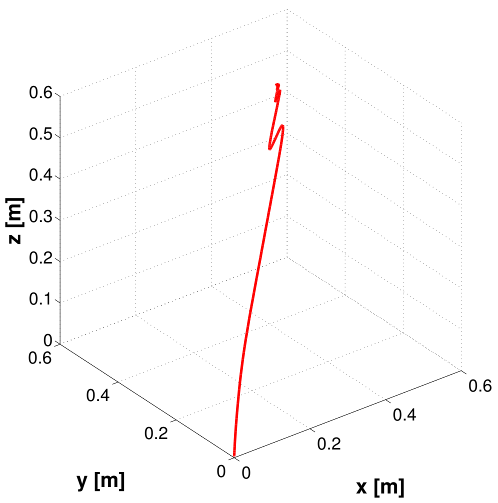
(a) x-position



(b) y-position



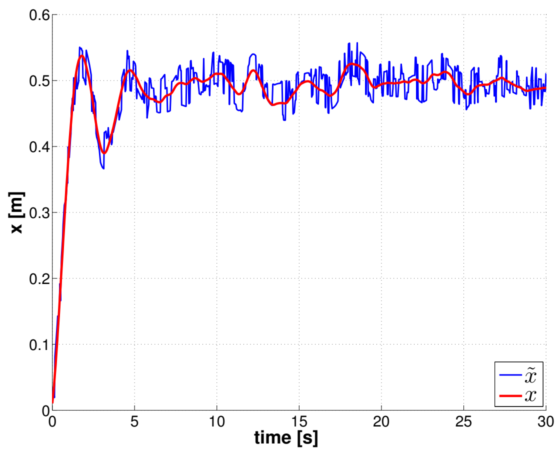
(c) z-position



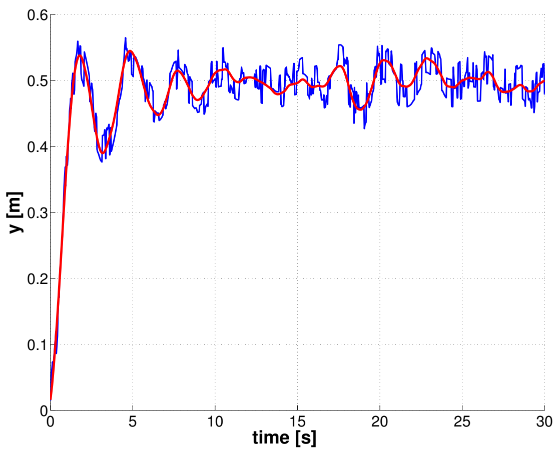
(d) 3D representation of the hover-trajectory

Figure 5.2: Hovering experiment without noise. After the approach phase, the *MAV* hovers at position $\mathbf{x}^* = [0.5\ 0.5\ 0.5]^T$. Figures (a)-(c) show the temporal evolution of the x, y and z coordinate whereas Figure (d) shows the spatial representation in 3D space.

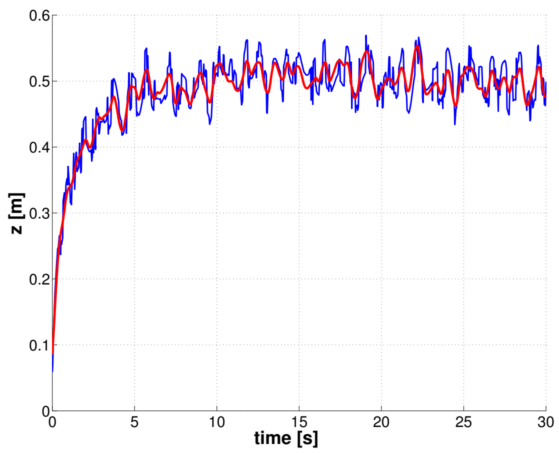
changing the noise level from one position update to the next. The resulting trajectories for x, y and z direction as well as the 3D trajectory are shown in Figure 5.3.



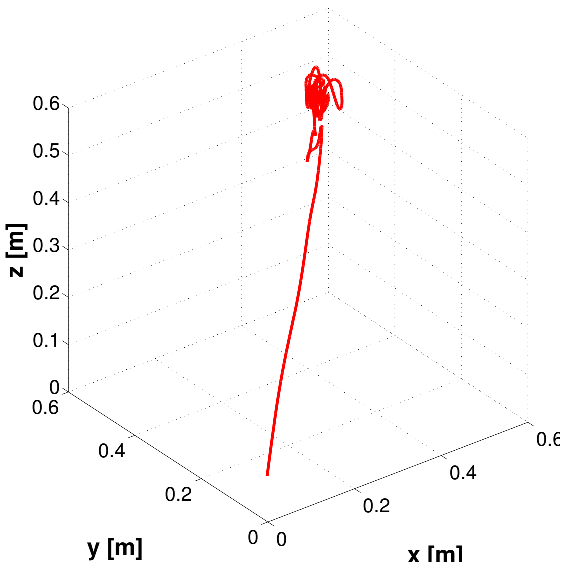
(a) x-position



(b) y-position



(c) z-position



(d) 3D representation of the hover-trajectory

Figure 5.3: Hovering experiment with noise. After the approach phase, the *MAV* hovers at position $\mathbf{x}^* = [0.5 \;\; 0.5 \;\; 0.5]^T$. Figures (a)-(c) show the temporal evolution of the x, y and z coordinate whereas Figure (d) shows the spatial representation in 3D space.

The red curve in Figure 5.3 represents the actual position of the *MAV*, whereas the blue curve represents the noisy position estimate provided as controller input. The noisy position estimate causes a greater deviation between the desired and the actual position compared to the noiseless case. This is also indicated by the computed *RMS*- and maximum error as shown in Table 5.3.

error [m]	z	xy	xyz
RMS	0.0185	0.0215	0.0284
max	0.0536	0.0482	0.0609

Table 5.3: Evaluation of hovering precision with noise. The *RMS*- as well as the maximum error are evaluated in different spaces: height z, horizontal xy-plane and 3D space xyz.

The *RMS* error is bound to 3 cm with a maximum error of about 6 cm. Note that the *RMS* error depicted in Table 5.3 is smaller than the chosen noise level of $\widehat{x} = 4\,\text{cm}$.

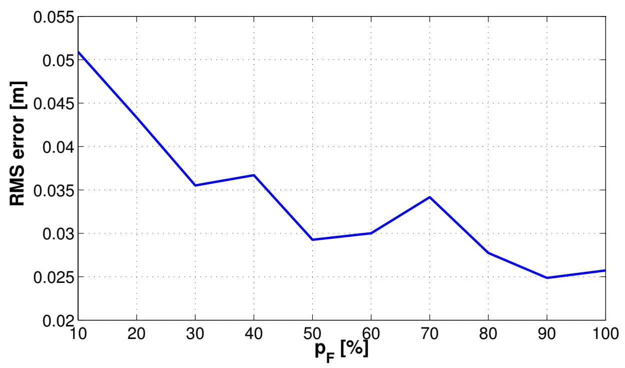


Figure 5.4: Impact of noise frequency: The *RMS* error decreases for increased noise frequency due to integrating controller behavior.

To further investigate the impact of the noise frequency, the *RMS* error for xyz was calculated for varying values of p_F. Low values of p_F lead to a slow variation of the noise level, whereas a high value leads to rapidly changing noise levels. The results of this analysis are summarized in Figure 5.4, where $\widehat{x} = 0.04$ m. For increasing frequency, the *RMS* error decreases due to the integrating controller behavior.

Finally, Figure 5.5 shows the resulting *RMS* error for xyz if both, the noise amplitude $\widehat{x}$ and the parameter p_F are varied. For this purpose, the *MAV* was hovering at the desired position and the *RMS* error was estimated over a period of $\tau = 30$ s for every combination of the noise parameters. Intuitively, the *RMS* error increases with increasing $\widehat{x}$, whereas the integrating controller can cope with high frequency noise, regardless of the actual noise level.

The simulations indicate that the controller design shows good robustness to noisy position estimates. In practice, the noise level depends on the distance between camera and feature points in the scene. For noise levels in the range of a few centimeters, the controller

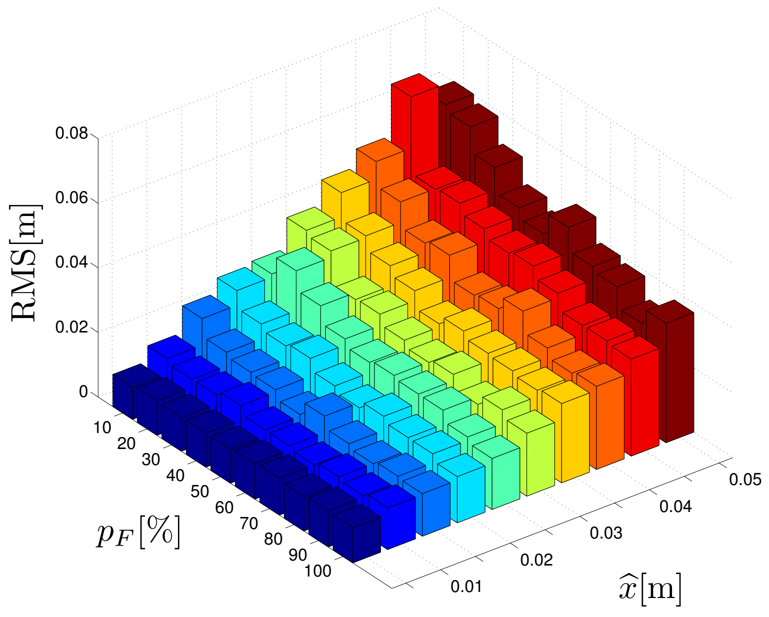


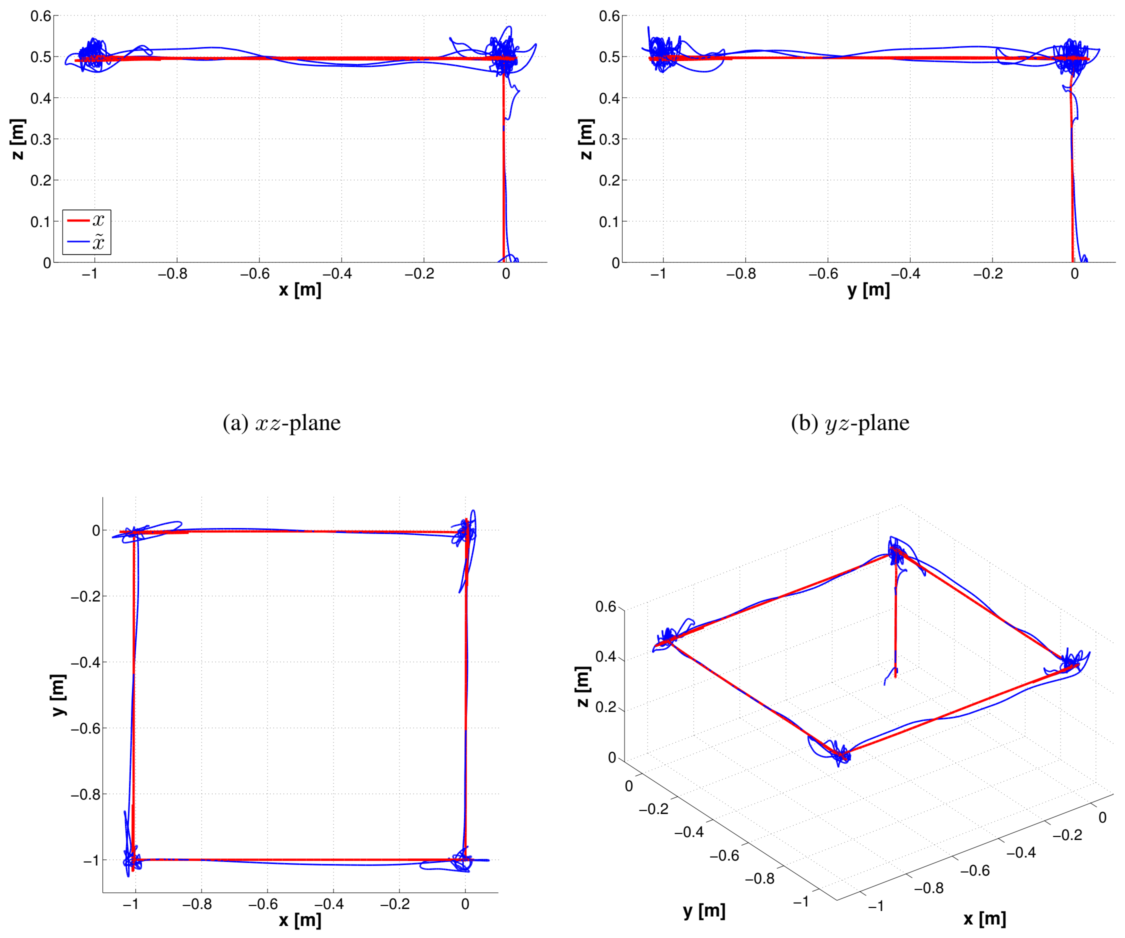
Figure 5.5: Noise evaluation: The *RMS* error decreases for increased noise frequency and increases with higher noise amplitude $\widehat{x}$.

can hold a given position with satisfying accuracy. In addition to the hovering experiment, the simulator was also used to analyze the performance while flying along a predefined trajectory.

5.2.2 Trajectory Flight

For this experiment, we define a square trajectory for the *MAV*. Starting from ground, the *MAV* flies along four waypoints, building a 1×1 m square in the horizontal plane, where every waypoint is approached in intervals of 30 seconds. The simulation is carried out for the noiseless case as well as for a noise level of $\widehat{x} = 0.04$ m and $p_F = 0.4$. The resulting trajectories are presented in Figure 5.6. The red curves in Figure 5.6 represent the trajectory of the *MAV* for the noiseless case, whereas the blue curve represents the trajectory when the position is corrupted by noise. Figure 5.6(a) - 5.6(c) show the projection of the trajectory onto the xz, yz and xy plane, respectively, and Figure 5.6(d) depicts the trajectory in 3D space. Whereas the trajectory of the *MAV* fits almost perfect to the desired trajectory for the noiseless case, there are slight deviations to the ideal path in presence of noise.

For the *RMS* error estimation, the path was sampled equidistantly along the trajectory in 0.5 mm steps. The resulting *RMS* error and the maximum error are presented in Table 5.4. The *RMS*



(a) xz-plane

(b) yz-plane

(c) xy-plane

(d) 3D representation

Figure 5.6: *MAV* trajectory along a predefined path for the noiseless (red) and noisy case (blue). Every 30 s, the *MAV* approaches the next waypoint.

error [m]	without noise	with noise
RMS	0.0085	0.0264
max	0.0499	0.0826

Table 5.4: Performance for the trajectory flight. The *RMS*- and maximum error for xyz are evaluated by sampling along the trajectory in 0.5 mm steps.

error is well below 3 cm for the noisy case and bound to a maximum of about 8 cm. This confirms the results from the hovering simulation and indicates that the controller design is suitable for operating with a noisy pose estimation.

The simulation results show that the design of the controller is robust to noisy pose estimates under realistic assumptions. In the next step, the performance of the controller is evaluated in an indoor– as well as outdoor–flight scenario using the Pelican quadrotor helicopter. As a tool for pose estimation, a real-time capable *VSLAM* approach is applied. This is discussed in the next section.

5.3 Visual SLAM for Pose Estimation

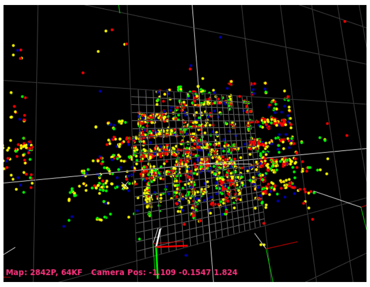
After having shown the robustness of the controller in a simulator, it is now moved to the Pelican quadrotor helicopter. There, a pose estimate has to be determined visually. Natural features present in the scene are used to create a sparse map of the environment in order to determine the location of the camera by applying the *VSLAM* algorithm of Klein and Murray [25] already described in Chapter 4. Their approach has been adapted to deliver a metrically correct map with known scale. This is achieved by modifying the initialization step: An artificial marker with known dimensions is used during the pose determination of the first two keyframes. Thereby the baseline between the first two cameras corresponds to the true metric distance between the first two keyframes and allows for a metrically correct initialization of the map. Moreover, the *MAV* is currently carried around by hand to expand the map in the local region where the *MAV* is going to take-off and land again. This ensures a proper pose estimate during take-off and is depicted in Figure 5.7. Additionally, the manually created initial map is extended online while the *MAV* explores the environment.

The quality of the map is very important in terms of pose estimation accuracy and thus a robust control of the *MAV*. The accuracy not only depends on the number of features but also on their distribution over the image. The more uniformly the features are distributed over the image, the better the pose estimate. Another important quantity is the number of keyframes. Whereas too little keyframes result in a rather inaccurate pose estimate, the time needed to administer the entire map rises with an increasing number of keyframes. Furthermore, the distance between camera and detected world features has an impact on the accuracy of the estimated pose. In general one could say that the farther away the features are from the camera, the poorer the pose estimate gets.

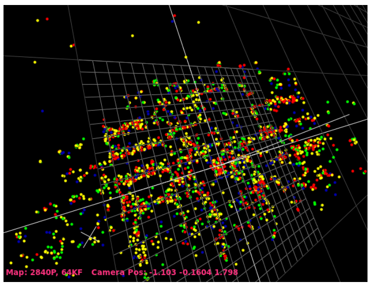
(a) Keyframe 1

(b) Keyframe 2

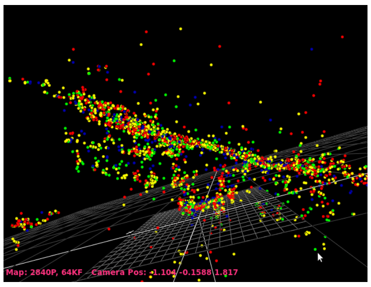
(c) Keyframe 3


Map: 2842P, 64KF Camera Pos: -1.109 -0.1547 1.824

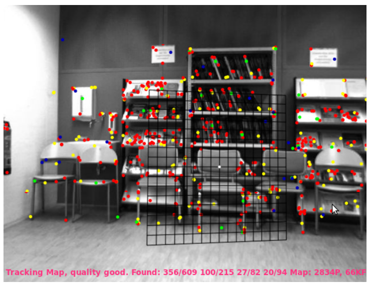
(d) Map front-view


Map: 2840P, 64KF Camera Pos: -1.103 -0.1604 1.798

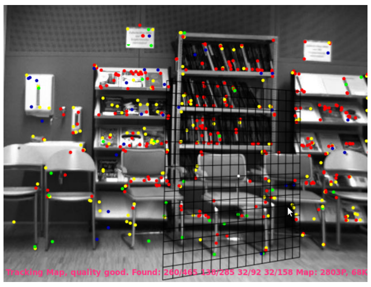
(e) Map side-view


Map: 2840P, 64KF Camera Pos: -1.104 -0.1588 1.817

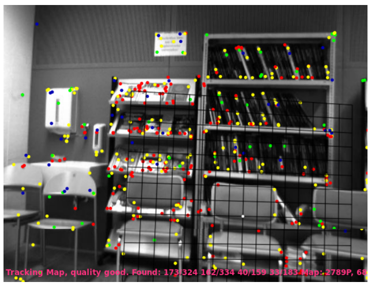
(f) Map top-view


Tracking Map, quality good. Found: 356/609 100/215 27/82 20/94 Map: 2834P, 66KF

(g) View 1


Tracking Map, quality good. Found: 260/465 [illegible]/285 32/92 32/158 Map: 2803P, 68K

(h) View 2


Tracking Map, quality good. Found: 173/324 162/334 40/159 33/183 Map: 2789P, 68

(i) View 3

Figure 5.7: *VSLAM* map extraction and pose estimation. Based on the keyframes depicted in Figures (a)–(c) a sparse map of the environment is estimated. Figures (d)–(f) show different views of the created map, and Figures (g)–(i) represent the re-detected map features in the current views of the *MAV*, which are then used to estimate the pose.

We have discussed how to determine a pose for real world experiments using a *VSLAM* approach, and we also mentioned some limitations this approach has towards a robust control of the *MAV* during take-off, landing and hovering. Now we are first going to analyze the performance of our approach in an indoor environment. Outdoor experiments will then be discussed afterwards.

5.4 Indoor Control

Indoors, we evaluate the precision of hovering as well as of trajectory-flights. Therefore, we compare the visual pose estimate to ground-truth data acquired from an off-the-self outside-in tracking system. Due to the limited amount of data that can be transferred from the *MAV* down to the ground-station via a wireless data link, we also evaluate the influence of reducing the camera's resolution from 640×512 *px* to 320×256 *px*. Additionally, we analyze the controller's reaction to inaccuracies of the pose estimate resulting from continually increasing the distance between the camera and detected world features. However, the last experiment is done without comparison to ground-truth as the tracking volume of the reference system is limited to $2 \times 2 \times 2$ m in our case, which is not sufficient to evaluate larger distances to the scene.

5.4.1 Comparison to Ground-Truth

First, we evaluate the accuracy of the vision-based pose estimate by comparison to ground-truth data. We use an outside-in tracking system from A.R.Tracking[1] delivering the high precision ground-truth of ± 1 mm. The system uses three *IR*-cameras to determine the exact location of a tracking target mounted on the *MAV* as depicted in Figure 5.8. For evaluation, the lower camera resolution of 320×256 *px* is used to determine inaccuracies in the worst case scenario. A map consisting of 53 keyframes and about 1000 map points is initially created and used for visual pose estimation. The mean distance of the 3D feature points to the *MAV* is approximately 3.5 m.

Hovering experiment. During this experiment the *MAV* is supposed to hover at a target position $\mathbf{x}^* = [0.0\ 0.0\ 1.5]^T$ for a period of $\tau = 30$ s. The results for this experiment are shown in Figure 5.9 for x, y and z coordinates respectively. The values are plotted over time after having reached the steady state position. Additionally, Figure 5.9(d) shows the hovering trajectory in 3D space. The blue curve represents the visually determined position used as controller input, while

[1] http://www.ar-tracking.de

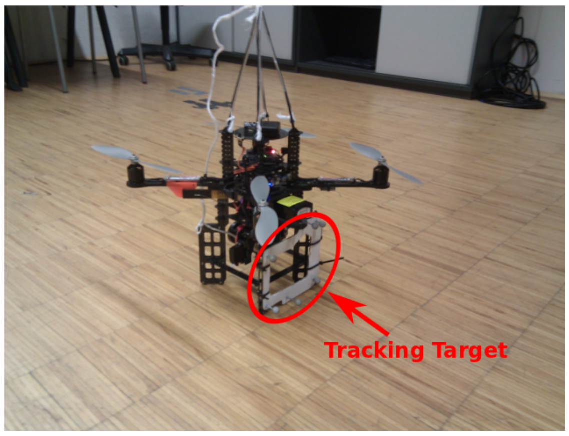


Figure 5.8: Tracking target used for pose estimation to acquire ground-truth using an outside-in tracking system.

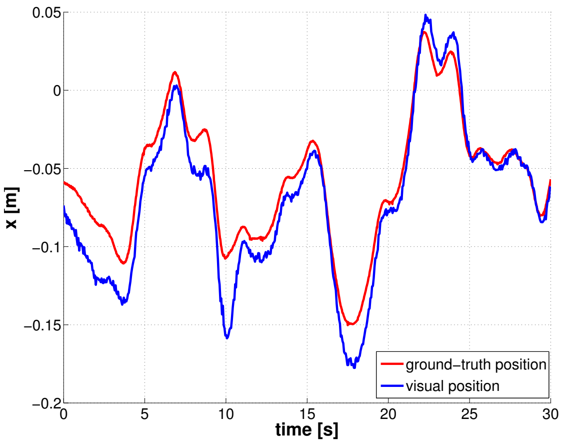
error [m]	320×256 *px*			ground-truth		
	z	xy	xyz	z	xy	xyz
RMS	0.0875	0.1413	0.1662	0.0825	0.1302	0.1541
max	0.1994	0.2711	0.3088	0.1802	0.2240	0.2652

Table 5.5: Hovering evaluation for the 320×256 *px* camera resolution and the ground-truth based on outside-in tracking. The *RMS* error of z as well as the error of the xy plane and the error in xyz space are evaluated.

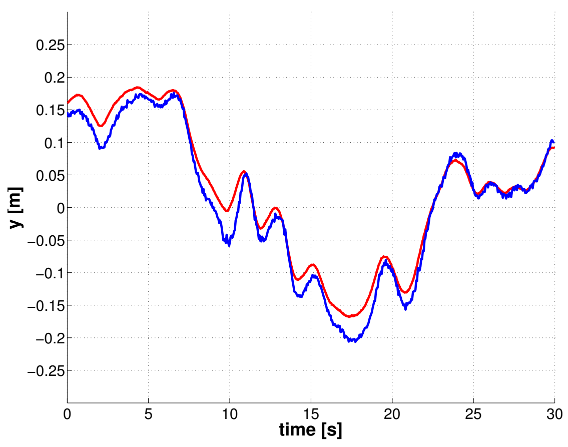
the red curve marks the ground-truth and thus the true position.
The results regarding *RMS*- and maximum error for this experiment are summarized in Table 5.5. The *RMS* error is bound to about 9 cm in height for the case where the position has been estimated visually, and to about 8 cm when looking at the ground-truth *RMS* error. Similar results can be observed when comparing the *RMS* error in the xy plane and in 3D space. Thus, the *RMS* error between visually determined positions and ground-truth positions is more or less the same, in other words the visually estimated position deviates only about ± 1 cm from ground-truth.

Compared to the simulated experiments, the *RMS* error is significantly higher. This is due to the fact that the simulator uses a simple dynamic model of the *MAV* while the true system is rather complex. Furthermore, in a real world environment, several disturbances occur, for example turbulences. These depend on the room's geometry and are thus not taken into account in the simulation.

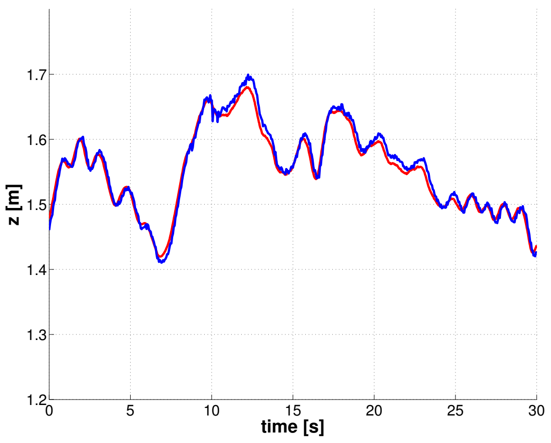
Trajectory flight experiment. In this experiment the *MAV* has to fly along a predefined trajectory consisting of four waypoints, resulting in a 0.5×0.5 m square in the horizontal plane at a



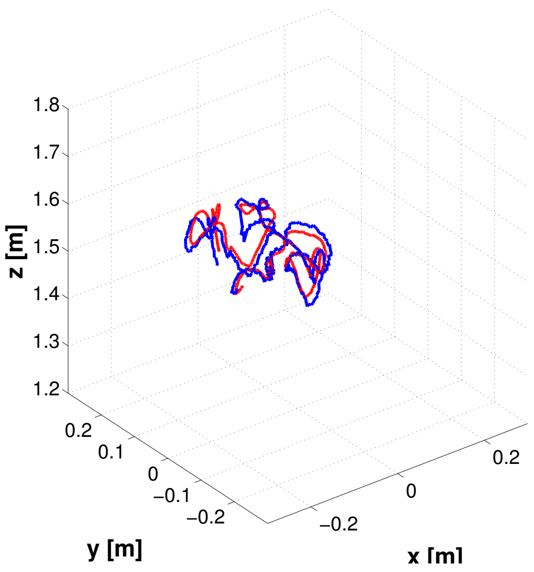
(a) x-position



(b) y-position



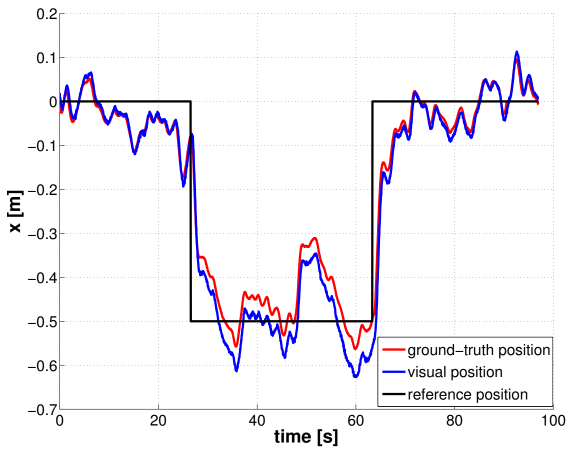
(c) z-position



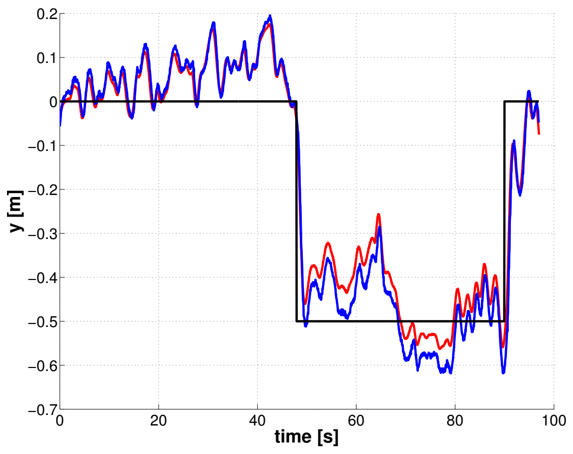
(d) 3D representation of the hover-trajectory

Figure 5.9: Hovering experiment based on an outside-in tracking ground-truth. The *MAV* hovers at position $\mathbf{x}^* = [0.0 \ \ 0.0 \ \ 1.5]^T$. Figures (a)-(c) show the temporal evolution of the x, y and z coordinate of the visual position (blue) as well as the *MAV* position based on the outside-in tracking (red), whereas Figure (d) shows the trajectory in 3D space.

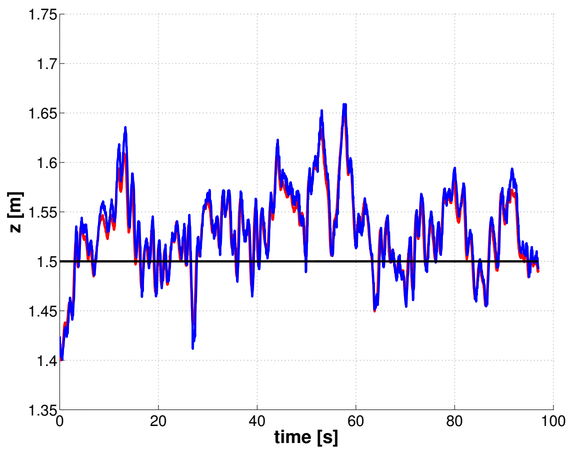
flying height of 1.5 m. The results of this experiment are depicted in Figure 5.10.



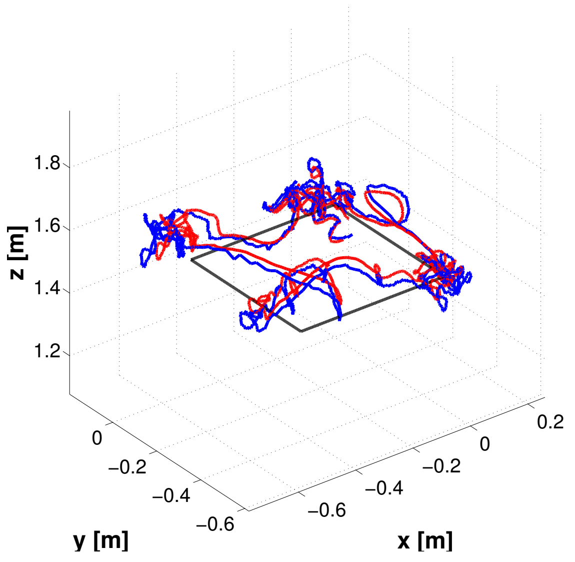
(a) x-position



(b) y-position



(c) z-position



(d) 3D representation of the trajectory flight

Figure 5.10: *MAV* trajectory flight along a predefined 0.5×0.5 m square path. Figures (a)-(c) show the temporal evolution of the x, y and z coordinate. Figure (d) shows the spatial representation of the trajectory in 3D space. The *RMS* error of the ground-truth to the predefined trajectory is 8.5 cm in 3D space.

Figures 5.10(a)–5.10(c) show the position of the *MAV* in x, y and z coordinate over time. Additionally, in Figure 5.10(d) the 3D space trajectory is depicted. The red curve shows the visually estimated position, the blue curve the ground-truth position and the black curve the manually defined path which the *MAV* should follow. Comparing the red and blue curve in the x-position plot of Figure 5.10(a) - which corresponds to the distance between *MAV* and map features - one can observe that the further away the *MAV* moves, the more inaccurate the visually estimated x-position gets compared to its true position.

Additionally, an *RMS* error evaluation has been performed, where the path has been subsampled equidistantly along the predefined trajectory in 0.5 mm steps. The ground-truth *RMS* error is 8.5 cm with a maximum error of 21.4 cm in 3D space, whereas the visually estimated position *RMS* error is 9.5 cm with a maximum error of 21.3 cm. Thus, during this experiment we could show that the visually acquired position estimate compared to ground-truth data is good enough to successfully control an *MAV*. Moreover, we could show that the distance between the *MAV* and the map influences the accuracy of the pose estimate.

5.4.2 Distance Evaluation

During the trajectory flight experiment we could observe that the visually acquired position estimate gets less accurate compared to ground-truth with increasing distance between *MAV* and map. Thus, further evaluate the robustness of our controller with respect to such inaccuracies of the visually estimated position by increasing this distance step by step as shown in Figure 5.11.

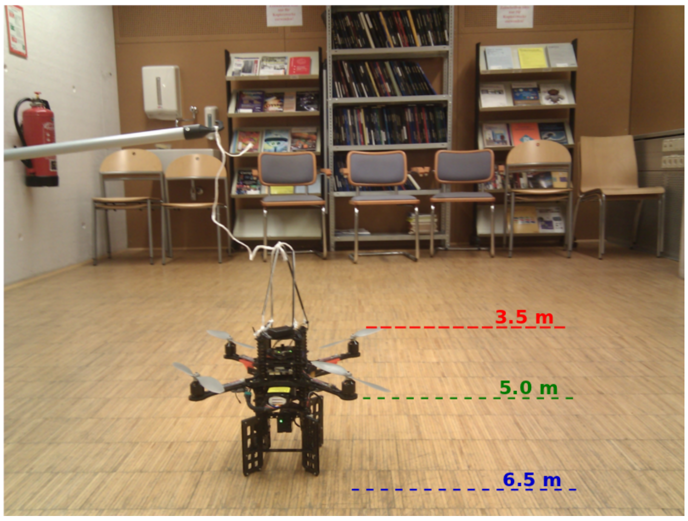


Figure 5.11: Three different distances to the scene, $3.5, 5.0$ and 6.5 m, are evaluated.

distance [m]	320×256 *px*			640×512 *px*		
	z	xy	xyz	z	xy	xyz
3.5	0.0626	0.1232	0.1394	0.0753	0.1218	0.1437
5.0	0.0520	0.1175	0.1287	0.0453	0.1138	0.1226
6.5	0.0554	0.1102	0.1234	0.0601	0.1516	0.1636

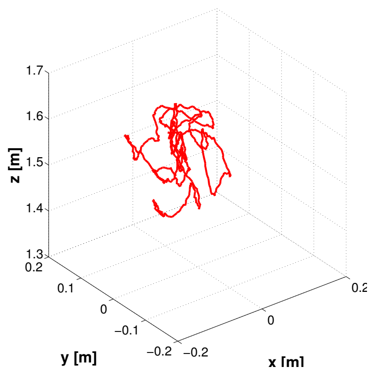
Table 5.6: The hovering *RMS* error, averaged over three flights, is evaluated based on different resolutions and distances to the natural features. The average *RMS* error of z as well as the average *RMS* error of the xy plane and the average *RMS* error in the three dimensional xyz space are presented.

Furthermore, we evaluate how a higher resolution of the camera affects the results. For the higher resolution of 640×512 *px* a map consisting of 64 keyframes and 2800 map points has been created, which is depicted in Figure 5.7. For the smaller resolution of 320×256 *px* the map consists of 51 keyframes and 1200 map points.

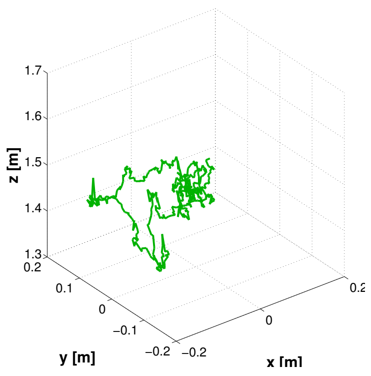
For all distances, namely $3.5\,\text{m}$, $5.0\,\text{m}$ and $6.5\,\text{m}$ and the two different image resolutions, we performed three hovering experiments and calculated the average *RMS*- and maximum error after the system reached the steady state at the desired position $\mathbf{x}^* = [0.0\ 0.0\ 1.5]^T$. The results are summarized in Table 5.6. For all experiments performed it seems that the distance does not affect the accuracy of the estimated position. However, we showed that this is the case when comparing to ground-truth. Thus, we had a closer look on the estimated position trajectories for the two different resolutions as depicted in Figure 5.12.

It can be observed for both resolutions that the higher the distance to the map gets, the nosier is the visual position estimate. The reason why there probably is no difference in accuracy between the two resolutions is that we had to compress the higher resolution image, whereas for the smaller image the raw image data could be transmitted. Thus, the compressed image suffers from compression artifacts which in turn affect the quality of the pose estimation process. Additionally, the trajectory is of course influenced by the controller, so a direct comparison of the pose estimates is difficult.

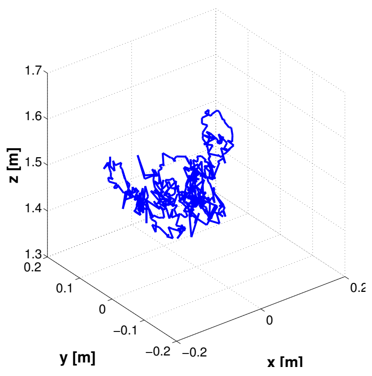
Finally, we compare our results to a related approach of Achtelik et al. [26]. We therefore picked the best of our hovering experiments and compared them to their results, as presented in Table 5.7. Best performance is achieved for both resolutions at a distance of $5.0\,\text{m}$, which is approximately in the middle of the room. We think that this is the best trial because the fewest turbulences occur. Achtelik et al. perform better than our approach in terms of keeping the correct height, which is due to fact that they use a camera looking down to the floor, whereas in



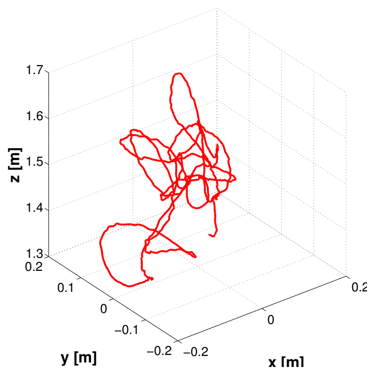
(a) 3.5m distance, 320×256 px



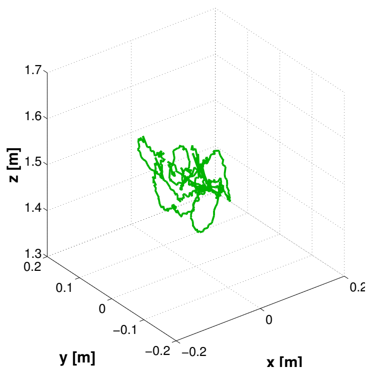
(b) 5.0m distance, 320×256 px



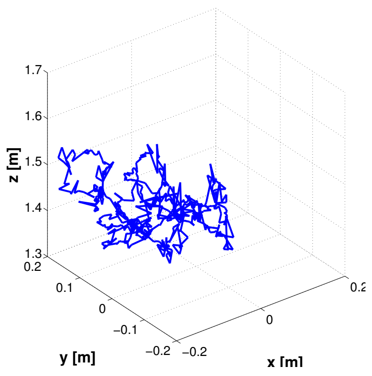
(c) 6.5m distance, 320×256 px



(d) 3.5m distance, 640×512 px



(e) 5.0m distance, 640×512 px



(f) 6.5m distance, 640×512 px

Figure 5.12: Indoor hovering evaluation for different scene depths and camera resolutions: with a desired hovering postion at $\mathbf{x}^* = [0.0\ 0.0\ 1.5]^T$. Figures (a)-(c) show the hovering trajectory for a 320×256 *px* camera resolution, whereas Figures (d)-(f) show the hovering trajectory for a 640×512 *px* camera resolution in 3D space.

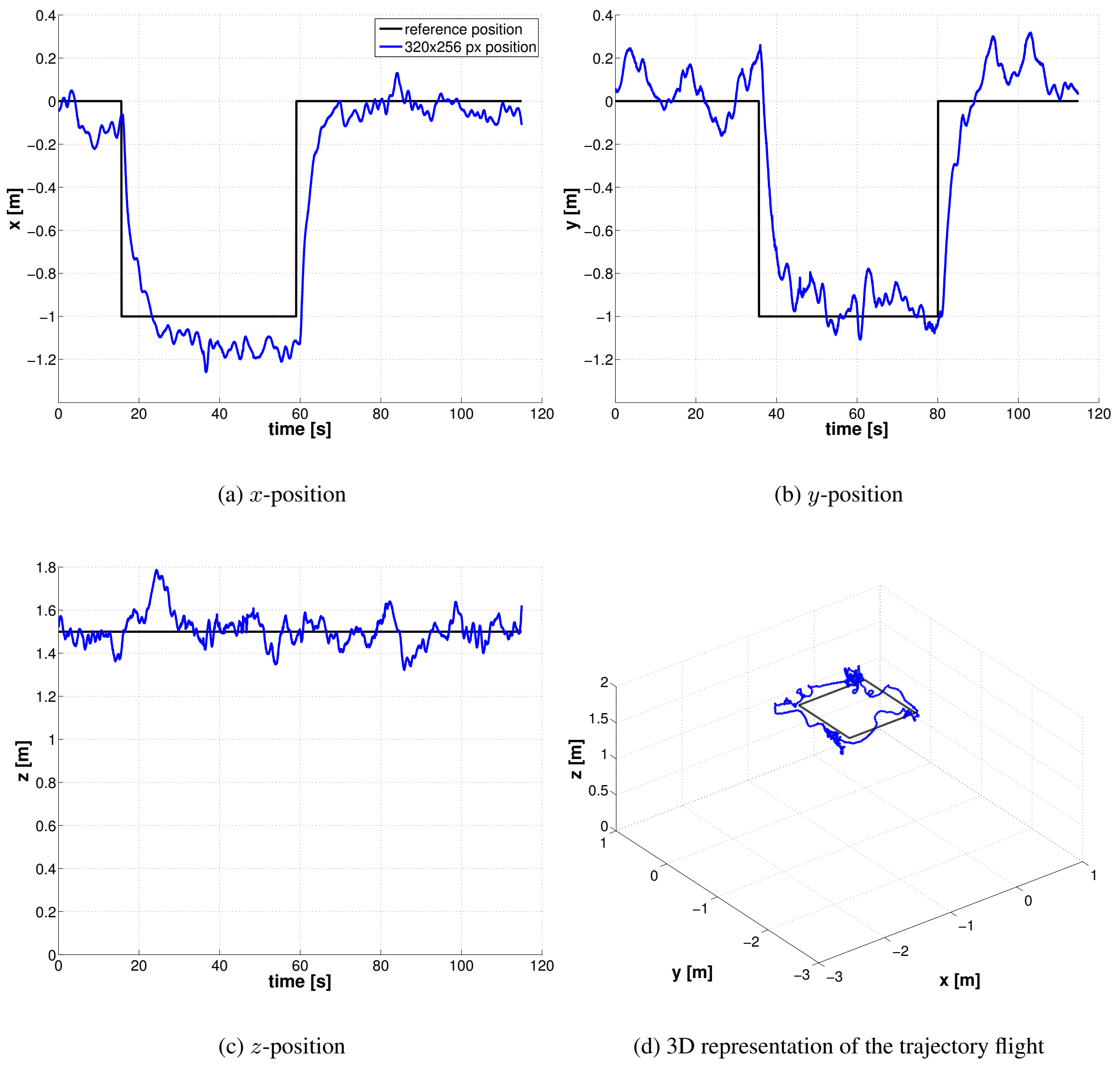
our case the camera does not. However, comparing the xy-error we outperform their approach with an *RMS* error of 6.7 cm compared to 12 cm. Unluckily, further comparison in the xyz-plane or regarding the maximum errors is not possible due to the missing evaluation in [26].

5.4.3 Trajectory Flights

A rather small trajectory flight of 0.5×0.5 m has already been evaluated and compared to ground-truth. Here, we evaluated another two trajectory flights, where we traveled a longer distance and evaluated the controller's behavior when using different image resolutions.

approach	distance [m]	*RMS* error [m]			max error [m]		
		z	xy	xyz	z	xy	xyz
ours, 320 × 256 *px*	3.5	0.0492	0.0947	0.1067	0.0984	0.1559	0.1620
	5.0	0.0512	0.0903	0.1038	0.1411	0.2067	0.2107
	6.5	0.0462	0.0914	0.1024	0.1147	0.1851	0.1865
ours, 640 × 512 *px*	3.5	0.0918	0.0953	0.1324	0.2408	0.1839	0.2767
	5.0	0.0485	0.0672	0.0828	0.1294	0.1259	0.1646
	6.5	0.0639	0.1318	0.1465	0.1868	0.2645	0.2647
Achtelik et al.[26]	1.4	0.01	0.12	–	–	–	–

Table 5.7: Indoor hovering evaluation of the *RMS* and maximum error over different scene depths and camera resolutions. The *RMS*- as well as the maximum error are evaluated in height z, xy-plane and the 3D space.



(a) x-position

(b) y-position

(c) z-position

(d) 3D representation of the trajectory flight

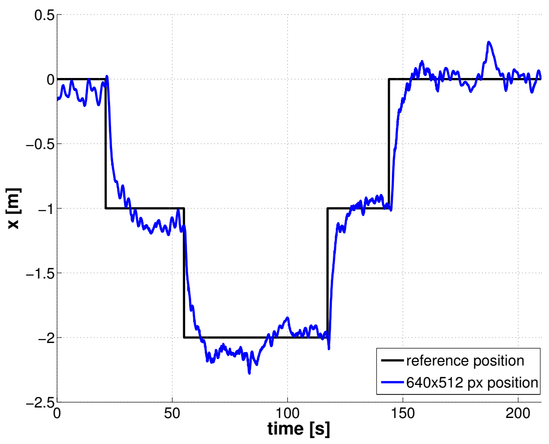
Figure 5.13: *MAV* trajectory along a predefined 1.0×1.0 m square path. Figures (a)-(c) show the temporal evolution of the x, y and z coordinate. Figure (d) shows the spatial representation of the trajectory in 3D space.

approach	x	y	z	xyz
our 320×256 *PTAM*	0.0704	0.1109	0.0663	0.1471
our 640×512 *PTAM*	0.0768	0.1104	0.0708	0.1520
Blösch et al. [1]	0.0995	0.0748	0.0423	–

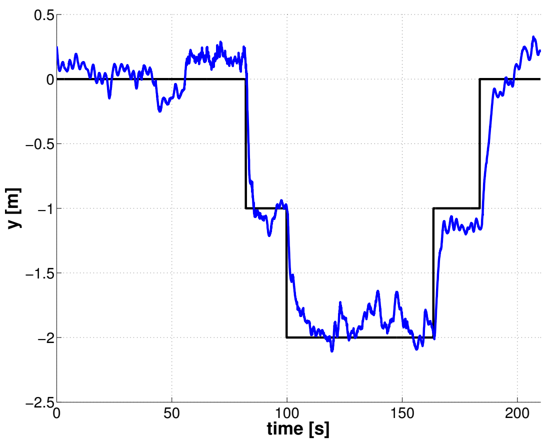
Table 5.8: Performance for the trajectory flight. The *RMS*- and maximum error for xyz are evaluated by sampling along the trajectory in 0.5 mm steps.

The experiments have been conducted with the same maps used during the distance evaluation. Figure 5.13 shows the trajectory flight using the 320×256 *px* camera resolution. The predefined path in the height of 1.5 m above ground followed a 1.0×1.0 m square which results in a total length of 4.0 m to travel. Figures 5.13(a)–5.13(c) show the temporal evolution of the flight path for the x-, y- and z-position, respectively. The black curve again marks the requested path. The 3D trajectory in space is depicted by Figure 5.13(d). The *RMS* error for this experiment is 15.06 cm in 3D space and is calculated by equidistantly sampling the predefined path in 0.5 mm steps.

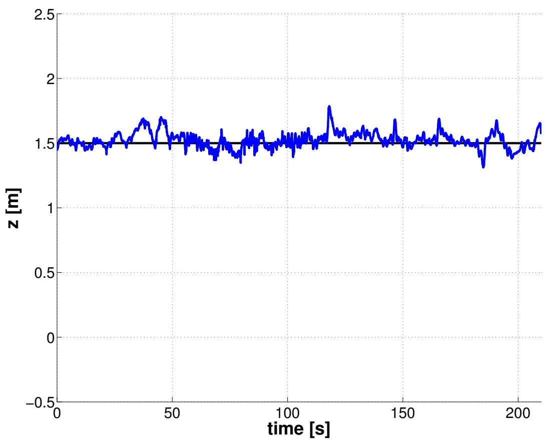
The results for a camera resolution of 640×512 *px* are shown in Figure 5.14. In this experiment we even followed a 2.0×2.0 m square flight path which results in a total path length of 8.0 m. The temporal evolution for the positions of x, y and z are again shown in Figure 5.14(a)–5.14(c), whereas the 3D trajectory is shown in Figure 5.14(d). The calculated *RMS* error is 14.71 cm in 3D space. To be able to compare our trajectory flights with others, we evaluated the *RMS* error for each coordinate separately. The results are depicted in Table 5.8 and are comparable to those of Blösch et al. [1]. However, our system does not rely on a textured ground plane therefore, we use the discriminative scene geometry naturally present. Additionally, our ground-station is not connected physically to the *MAV* which is important in terms of autonomous inspection tasks. Furthermore, our approach does not require a mathematical model of the *MAV* for control design in comparison to [1].



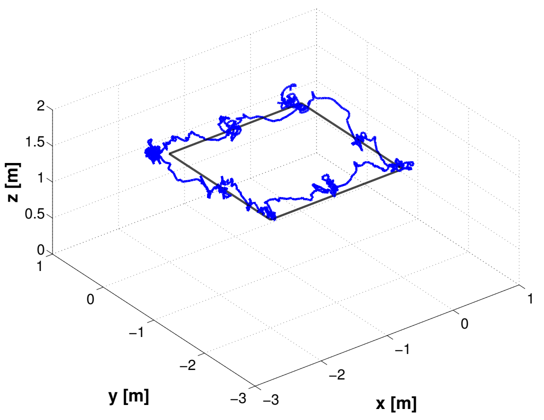
(a) x-position



(b) y-position



(c) z-position



(d) 3D representation of the trajectory flight

Figure 5.14: *MAV* trajectory along a predefined 2.0×2.0 m square path. Figures (a)-(c) show the temporal evolution of the x, y and z coordinate. Figure (d) shows the spatial representation of the trajectory in 3D space.

5.5 Outdoor Control

After having evaluated several experiments for indoor environments we want to demonstrate the applicability to an outdoor environment as well. We used the lower resolution of 320×256 *px*, which allows real-time transmission of the images with 30 *FPS* without compression, and created a map consisting of 46 keyframes and 560 map points. We hovered at a target position $\mathbf{x}^* = [0.0\ 0.0\ 1.5]^T$ for a period of $\tau = 30$ s in front of a building with little associated texture, as depicted in Figure 5.15.

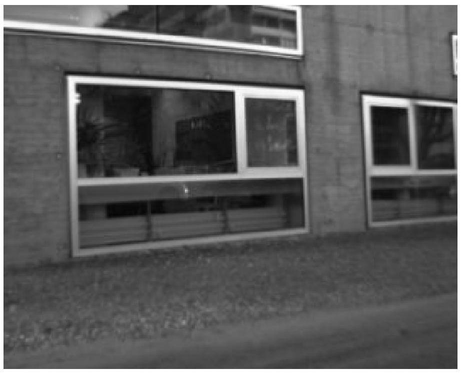
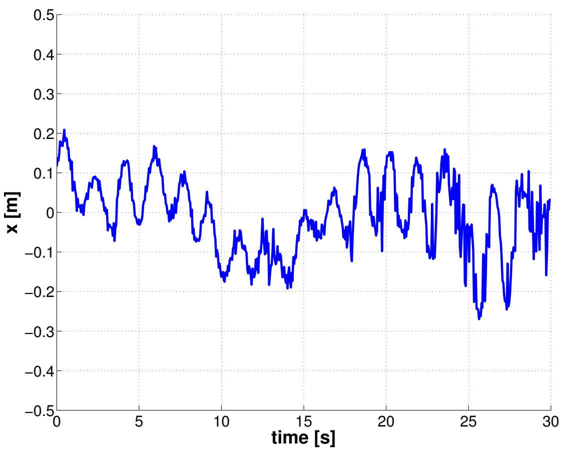


Figure 5.15: Outdoor evaluation scene in front of a building. Note that in this experiment only little texture is available for the pose estimation.

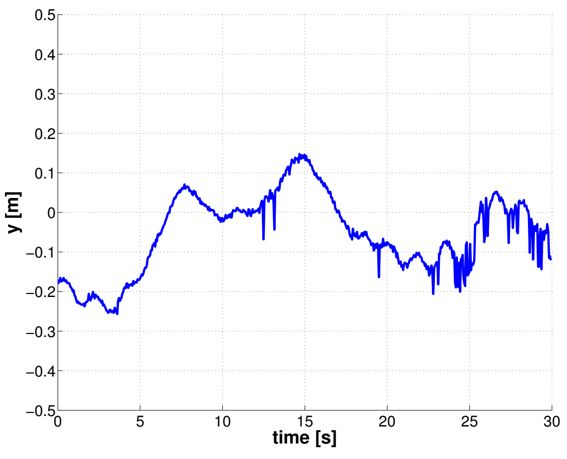
approach	distance [m]	*RMS* error [m]			max error [m]		
		z	xy	xyz	z	xy	xyz
ours, 320×256 *px*	10.0	0.0643	0.1495	0.1627	0.2189	0.2716	0.3471
Achtelik et al.[26]	3.3	0.11	0.44	–	–	–	–

Table 5.9: Performance of the outdoor hovering. The *RMS*- and maximum error for z, xy-plane and in 3D space xyz are shown and compared with the approach of Achtelik et al. [26].

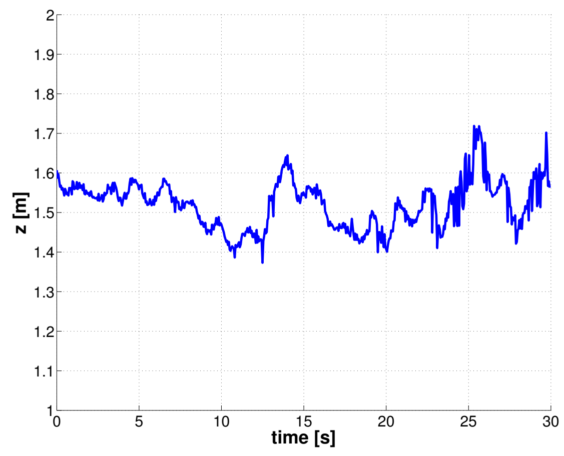
The distance to the building was approximately 10 m, and the experiment took place on a day with good wind conditions. The resulting x-, y-, and z-positions are visualized in Figure 5.16(a)–5.16(c), whereas the 3D trajectory can be seen in Figure 5.16(d). The results are summarized and compared to those of Achtelik et al. [26] in Table 5.9. As can be seen from the results, we clearly outperform their approach in an outdoor setting with an *RMS* error of 15 cm compared to 44 cm in the xy-plane. However, we also want to note that it is difficult to compare the results as the experiment took place outdoors and the results highly depend on the scene and on the wind conditions.



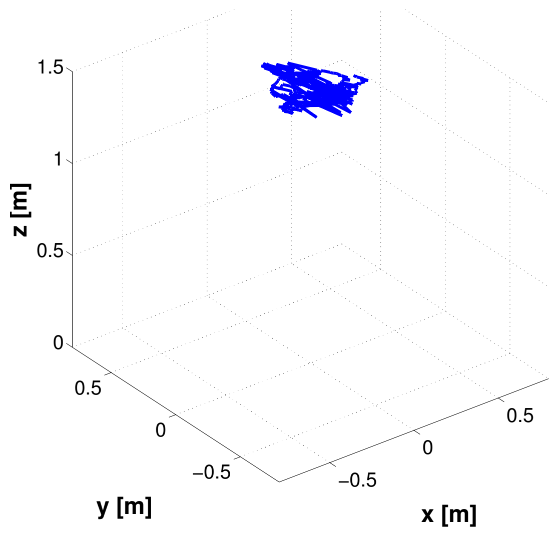
(a) x-position



(b) y-position



(c) z-position

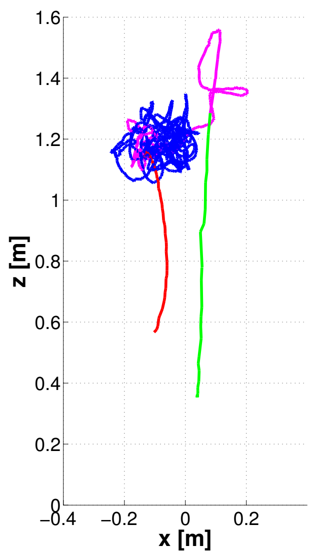


(d) 3D representation of the outdoor hover-trajectory

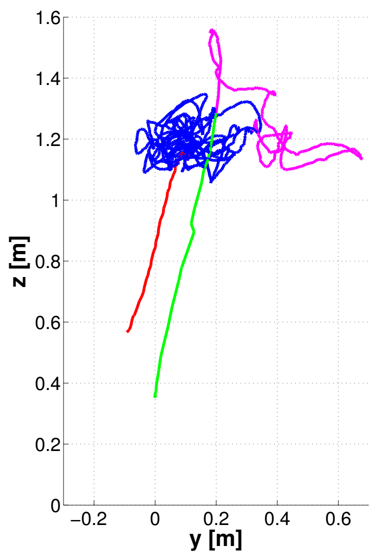
Figure 5.16: Outdoor hovering experiment: The *MAV* hovers at position $\mathbf{x}^* = [0.0 \;\; 0.0 \;\; 1.5]^T$. Figures (a)-(c) show the temporal evolution of the x, y and z coordinate whereas Figure (d) shows the trajectory in 3D space.

5.6 A Typical Take-Off, Hovering, and Landing Example

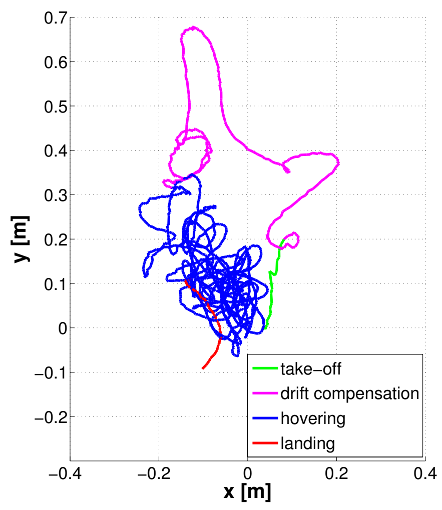
This section shows a typical example for autonomous take-off, hovering and landing of an *MAV* as performed many times. Figure 5.17 illustrates four different states, including drift compensation which is needed to overcome the typical offsets in the dynamic range of the input. In other words, the integration subsystems of the controller automatically estimate the best parameters for efficient control.



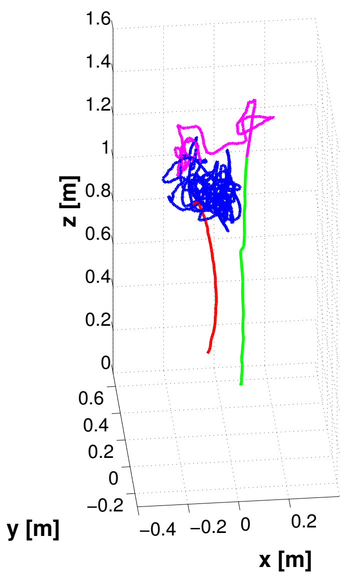
(a) xz-plane



(b) yz-plane



(c) xy-plane



(d) 3D representation

Figure 5.17: *MAV* trajectory during take-off (green), drift compensation (magenta), hovering (blue) and landing (red).

During take-off, marked by the green trajectory, the almost linear behavior is clearly visible. The

magenta curve marks the drift compensation, whereas the blue curve shows the *MAV* hovering trajectory at a desired height of 1.2 m. Finally, the red trajectory shows the landing of the *MAV*, where tracking is lost shortly before reaching the floor. Nevertheless, safe landing was always possible within a radius of about 0.5 m compared to the commanded hovering position, such as depicted in Figure 5.17.
To support this example, a video showing the previously described behavior is available online[2].

5.7 Discussion

During indoor and outdoor hovering, as well as several trajectory flights, we have shown that our approach is suitable for the tasks of automatic take-off, hovering, and landing of *MAV*s. We only use visual input to determine a pose and are even capable of dealing with quite noisy pose estimates. We evaluated our approach in terms of accuracy by comparison to ground-truth data, and conducted several experiments, including different distances and camera resolutions. During all experiments we were able to autonomously take-off, perform the desired task of hovering or following a predefined path, and then land again within a range of approximately 0.5 m from the desired landing location. Additionally, our approach is able to detect system failures and react accordingly. For example, when no valid pose estimate is available, the *MAV* starts the automatic landing procedure or switches to *GPS* position hold mode in outdoor flights.

The benefit of the presented approach is that no mathematical model of the *MAV* is required. Thus, we are flexible in terms of the configuration setup, since the controller is not optimized to a specific payload. Additionally, the system is robust to noisy pose estimates and shows comparable performance to other state-of-the-art systems that combine several sensor modalities. One drawback that is inherent to all vision-based servoing systems is that the approach is limited to structured scenes and that the performance depends on the illumination conditions. We plan to tackle this issue in future work by using a High Dynamic Range (*HDR*) camera that can deal with lighting variations.

[2]http://aerial.icg.tugraz.at

Chapter 6

Summary

Contents

In this work we have presented a human-inspired visual servoing approach for automatic take-off, hovering, and landing of *MAV*s which is suitable for outdoor as well as for indoor applications. A monocular camera is used as an input sensor to robustly control the *MAV*. To conclude, we give a summary of our contributions and an outlook to future work.

6.1 Conclusion

We have shown that we are able to robustly control an *MAV* in indoor as well as in outdoor environments based on visually detected natural features. During a flight, the pose of the *MAV* is estimated using a camera which looks forwards. To keep track of the camera, the environment is sparsely reconstructed. In contrast to a camera facing the floor, we look at the typically well structured scene and are thus able to estimate a pose soon after take-off.

During trajectory-flights we have shown that our approach is able to estimate a pose over several scales. Moreover, the visual pose estimation can handle partial occlusions in form of moving objects within the scene if enough static map features are still visible in the image. Furthermore, we showed that even a low camera resolution delivers a pose estimate accurate enough for autonomous navigation tasks.

Our approach is based on a *PBVS* technique which has the major advantage that the pose estimation and the controller are independent and can be replaced easily. We have shown that the fuzzy logic controller yields good precision comparable to other state-of-the-art approaches, not only for hovering tasks but as well for trajectory flights, without using a mathematical model of the *MAV*. Furthermore, we have demonstrated that our fuzzy logic controller is robust to noisy pose estimates, even without incorporating other sensor measurements. Additionally, we discussed how to detect system failures and how to react in indoor as well as in outdoor environments. Our system is realized as a *ROS*-node, which has the benefit that individual parts of our system can be distributed over several computers.

The presented system shows a robust behavior in in- and outdoor experiments and thus forms a reliable basis for the autonomous inspection of overhead power lines and power pylons targeted in the *PEGASUS* project, where autonomous take-off and landing are essential. Since no mathematical model of the *MAV* is required for the controller, the proposed system can be extended by further sensors to assist the inspection task.

6.2 Future Work

Our implementation for autonomous navigation of *MAV*s is solely based on visual input. However, most *MAV*s have additional sensors like *GPS*, air-pressure, compass and *IMU* which can also be used for pose estimation. Therefore, these sensor data and the visual data can be fused to achieve more precise pose estimates.

Another interesting topic for further research is learning the boundaries of the membership functions of the fuzzy logic. For example, based on the mean distance to natural features, the ϵ value which defines the noise level could be updated automatically.

Due to the bottleneck of the wireless connection and the low computational power available to the quadrotor helicopter, the tracking and mapping tasks of the visual *SLAM* approach can be split apart, such that tracking is performed directly on the *MAV* and mapping can be done on a more powerful ground-station. A lower resolution image could then be streamed down to the ground-station and used as a preview, whereas higher resolution images of newly added keyframes could be transferred to the ground-station to extend the map. Then, the new map can be transmitted back to the *MAV* for pose tracking.

Another interesting extension of this work would be to combine our system with a path planing algorithm. For this purpose, the next waypoint within a planned path is approached autonomously using our framework. The path planing requires the sparse map to represent obstacles, for example by means of a safety distance in order to avoid collisions. This is especially important for the inspection of power lines and pylons in the *PEGASUS* project to avoid an electric flash-over.

Appendix A

Acronyms

AR	Augmented Reality
AscTec	Ascending Technologies GmbH
BIBO	Bounded Input Bounded Output
CAD	Computer-Aided Design
CMOS	Complementary Metal Oxide Semiconductor
CoA	Center-of-Area
CoG	Center-of-Gravity
CoM	Center-of-Maximum
ECEF	Earth-Centered, Earth-Fixed
EKF	Extended Kalman Filter
FAST	Features from Accelerated Segment Test
FCU	Flight Control Unit
FFG	Austrian Research Promotion Agency
FPS	Frames Per Second
GPS	Global Positioning System
HDR	High Dynamic Range
HLP	High Level Processor
IBVS	Image-Based Visual Servoing
IMU	Inertial Measurement Unit
IR	Infrared
LLP	Low Level Processor
LQG/LTR	Linear Quadratic Gaussian Control Design with Loop Transfer Recovery
LRF	Laser Range Finder

MAV	Micro Aerial Vehicle
MIMO	Multiple Input Multiple Output
MISO	Multiple Input Single Output
MoM	Mean-of-Maximum
PBVS	Position-Based Visual Servoing
PID	proportional, integral and differential
PTAM	Parallel Tracking and Mapping
P3P	Perspective-3-Point-Problem
px	Pixel
RC	Remote Control
RGB-D	Red-Green-Blue-Depth
RMS	Root Mean Square
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
UAV	Unmanned Aerial Vehicle
VTOL	vertical take-off and landing
VSLAM	Visual Simultaneous Localization and Mapping
WiFi	Wireless Fidelity

Bibliography

[1] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart, “Vision Based MAV Navigation in Unknown and Unstructured Environments,” in *Proceedings of International Conference on Robotics and Automation*, (Anchorage, Alaska), pp. 21–28, 2010.

[2] S. Saripalli, J. F. Montgomery, and G. S. Sukhatme, “Vision-based Autonomous Landing of an Unmanned Aerial Vehicle,” in *Proceedings of International Conference on Robotics and Automation*, (Washington DC, US), pp. 2799–2804, 2002.

[3] B. Siciliano and O. Khatib, eds., *Springer Handbook of Robotics.* Springer, 2008.

[4] L. Mejias, P. Campoy, S. Saripalli, and G. S. Sukhatme, “A Visual Servoing Approach for Tracking Features in Urban Areas using an Autonomous Helicopter,” in *Proceedings of International Conference on Robotics and Automation*, (Orlando, US), pp. 2503–2508, 2006.

[5] C. Teulière, L. Eck, E. March, and N. Guénard, “3D model-based tracking for UAV position control,” in *Proceedings of International Conference on Intelligent Robots and Systems*, (Taipei, TW), pp. 1084–1089, October 2010.

[6] C. Whitworth, A. Duller, D. Jones, and G. Earp, “Aerial video inspection of overhead power lines,” *Power Engineering Journal*, vol. 15, no. 1, pp. 25–32, 2001.

[7] C. Steger, M. Ulrich, and C. Weidemann, *Machine Vision Algorithms and Applications.* Weinheim, DE: Wiley, 6th ed., 2008.

[8] A. R. Conway, *Autonomous Control of an Unstable Model Helicopter Using Carrier Phase GPS Only.* PhD thesis, University of Stanford, 1995.

[9] G. Hoffmann, D. G. Rajnarayan, S. L. Waslander, D. Dostal, J. S. Jang, and C. J. Tomlin, “The Stanford testbed of autonomous rotorcraft for multi agent control (STARMAC),” in *Proceedings 23rd Digital Avionics Systems Conference*, vol. 2, (Salt Lake City, UT), pp. 121–131, 2004.

[10] R. He, S. Prentice, and N. Roy, “Planning in Information Space for a Quadrotor Helicopter in a GPS-denied Environment,” in *Proceedings of International Conference on Robotics and Automation*, (Pasadena, CA), pp. 1814–1820, 2008.

[11] D. H. Shim, H. Chung, H. J. Kim, and S. Sastry, “Autonomous Exploration in Unknown Urban Environments for Unmanned Aerial Vehicles,” in *Proceedings AIAA Guidance, Navigation, and Control*, (San Francisco, CA), pp. 1–8, 2005.

[12] A. Bachrach, S. Prentice, R. He, and N. Roy, “RANGE - Robust Autonomous Navigation in GPS-denied Environments,” *Journal of Field Robotics*, vol. 28, no. 5, pp. 644–666, 2011.

[13] N. Karlsson, E. D. Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. E. Munich, “The vSLAM Algorithm for Robust Localization and Mapping,” in *Proceedings of International Conference on Robotics and Automation*, (Barcelona, ES), pp. 24–29, 2005.

[14] J. Stowers, M. Hayes, and A. Bainbridge-Smith, “Altitude Control of a Quadrotor Helicopter Using Depth Map from Microsoft Kinect Sensor,” in *Proceedings of IEEE International Conference on Mechatronics*, (Istanbul, TR), pp. 358 –362, 2011.

[15] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, “Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera,” in *Proceedings of the International Symposium of Robotics Research*, (Flagstaff, AZ), 2011.

[16] F. Chaumette and S. Hutchinson, “Visual Servo Control Part I. Basic Approaches,” *IEEE Robotics and Automation Magazine*, pp. 82 –90, December 2006.

[17] S. Hutchinson, G. D. Hager, and P. I. Corke, “A Tutorial on Visual Servo Control,” *IEEE Transactions on Robotics and Automation*, pp. 651–670, October 1996.

[18] J. Hill and W. Park, “Real Time Control of a Robot with a Mobile Camera,” in *Proceedings 9th International Symposium on Industrial Robots*, (Washington, DC), pp. 233–246, 1979.

[19] J. R. Azinheira and P. Rives, “Image-Based Visual Servoing for Vanishing Features and Ground Lines Tracking: Application to a UAV Automatic Landing,” *International Journal of Optomechatronics*, pp. 275–295, 2008.

[20] B. D. Lucas and T. Kanade, “An Iterative Image Registration Technique with an Application to Stereo Vision,” in *Proceedings 7th international joint conference on Artificial intelligence*, (Vancouver, CA), pp. 674–679, 1981.

[21] C. Tomasi and T. Kanade, “Shape and motion from image streams: a factorization method,” tech. rep., International Journal of Computer Vision, 1991.

[22] T. F. Goncalves, J. R. Azinheira, and P. Rives, “Homography-Based Visual Servoing of an Aircraft for Automatic Approach and Landing,” in *Proceedings of International Conference on Robotics and Automation*, (Anchorage, Alaska), pp. 9–14, 2010.

[23] C. S. Sharp, O. Shakernia, and S. Sastry, "A Vision System for Landing an Unmanned Aerial Vehicle.," in *Proceedings of International Conference on Robotics and Automation*, (Seoul, KR), pp. 1720–1727, 2001.

[24] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal Of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[25] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," in *Proceedings 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, (Nara, JP), pp. 225–234, November 2007.

[26] 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 *Proceedings of International Conference on Robotics and Automation*, (Shanghai, CN), 2011.

[27] R. Szeliski, *Computer Vision: Algorithms and Applications.* Springer, 2011.

[28] H. J. Zimmermann, *Fuzzy Set Theory and its Applications.* Boston, MA: Kluwer Academic Publishers, 1991.

[29] R. M. Haralick, C. Lee, K. Ottenberg, and M. Nölle, "Analysis and Solutions of the Three Point Perspective Pose Estimation Problem," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (Maui, HI), pp. 592–598, 1991.

[30] J. Rekimoto, "Matrix: A Realitime Object Identification and Registration Method for Augmented Reality," in *Proceedings of Asia Pacific Computer-Human Interaction*, (Kangawa, JP), pp. 63–68, 1998.

[31] H. Kato and M. Billinghurst, "Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System," in *Proceedings 2nd International Workshop on Augmented Reality*, (San Francisco, CA), pp. 85–94, 1999.

[32] D. Wagner and D. Schmalstieg, "ARToolKitPlus for Pose Tracking on Mobile Devices," in *Proceedings 12th Computer Vision Winter Workshop*, (St. Lambrecht, AT), pp. 139–146, 2007.

[33] G. Schweighofer and A. Pinz, "Robust Pose Estimation from a Planar Target," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2024–2030, 2006.

[34] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem,” in *Proceedings of the AAAI National Conference on Artificial Intelligence*, (Edmonton, CA), pp. 593–598, 2002.

[35] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges,” in *Proceedings 6th International Joint Conference on Artificial Intelligence*, (Acapulco, MX), pp. 1151–1156, 2003.

[36] G. Welch and G. Bishop, “An Introduction to the Kalman Filter,” tech. rep., University of North Carolina at Chapel Hill, 1995.

[37] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking,” *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[38] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, “MonoSLAM: Real-time single camera SLAM,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 1052–1067, 2007.

[39] M. Horn and N. Dourdoumas, *Regelungstechnik*. München, DE: Pearson Studium, 2004.

[40] J. Doyle, B. Francis, and A. Tannenbaum, *Feedback Control Theory*. Macmillan Coll Div, 1990.

[41] E. D. Sontag, *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. New York,US: Springer, second ed., 1998.

[42] K. W. Weng and M. S. B. Abidin, “Design and Control of a Quad-Rotor Flying Robot For Aerial Surveillance,” in *Proceedings 4th Student Conference on Research and Development*, (Selangor, MY), pp. 173 –177, 2006.

[43] J. Kim, M.-S. Kang, and S. Park, “Accurate Modeling and Robust Hovering Control for a Quad–rotor VTOL Aircraft,” *Journal of Intellignet & Robotic Systems*, vol. 57, no. 1-4, pp. 9–26, 2010.

[44] K. Watanabe, Y. Yoshihata, Y. Iwatani, and K. Hashimoto, “Image-Based Visual PID Control of a Micro Helicopter Using a Stationary Camera,” *Advanced Robotics*, vol. 22, no. 2-3, pp. 381–393, 2008.

[45] L. A. Zadeh, “Fuzzy Sets,” *Information and Control*, vol. 8, no. 3, pp. 338–353, 1965.

[46] I. S. Shawm, *Fuzzy Control of Industrial Systems: Theory and Applications.* Springer, 1998.

[47] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: an open-source robot operating system,” in *Proceedings on Open-Source Software workshop at the International Conference on Robotics and Automation*, (Kobe, JP), 2009.

[48] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” in *Proceedings 9th European Conference on Computer Vision*, (Graz, AT), pp. 430–443, 2006.

[49] P. J. Huber, “Robust Estimation of a Location Parameter,” *Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, 1964.

[50] V. Varadarajan, *Lie Groups, Lie Algebras and Their Representations.* New York, US: Springer Verlag, 1984.

[51] A. Leick, *GPS Satellite Surveying 2/e.* John Wiley & Sons, Inc., 1995.

[52] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, “The GRASP Multiple Micro UAV Testbed,” *IEEE Robotics and Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.

[53] N. Johnson, S. Kotz, and A. Kemp, *Univariate Discrete Distributions.* New York, US: Wiley, second ed., 1993.