Velocity controller design for a fly-by-wireless quadcopter

Master-Thesis
Tobias Tüylü

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich ETiT
Institut für Automatisierungstechnik
Prof. Dr.-Ing. Jürgen Adamy

Velocity controller design for a fly-by-wireless quadcopter
Master-Thesis

Eingereicht von Tobias Tüylü
Tag der Einreichung: 15. August 2016

Gutachter: Prof. Dr.-Ing. Jürgen Adamy
Betreuer: Raul Godoy

Technische Universität Darmstadt
Fachbereich ETiT
Institut für Automatisierungstechnik
Prof. Dr.-Ing. Jürgen Adamy

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in dieser oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 15. August 2016 Tobias Tüylü

Contents

List of Figures

Acronym

Acronym	
GPS	Global Positioning System
IMU	Inertial Measurement Unit
SDK	Software Development Kit
UAV	Unmanned Aerial Vehicle
PTAM	Parallel Tracking and Mapping
ROS	Robot Operating System
REP	ROS Enhancement Proposal
osrf	Open Source Robotics Foundation
DC	Direct Current
LSE	Least Square Estimation
KF	Kalman Filter
EKF	Extended Kalman Filter
NCS	Networked Control System
TDS	Time Delayed Systems
OP-A	Observer-Predictor Algorithm

State variables

$\mathbf{x}$	State vector
x,y,z	Translational movement
θ,ϕ,ψ	Pitch angle (y-axis), roll angle (x-axis), yaw angle (z-axis)
$\mathbf{R}$	Rotation matrix
$\mathbf{x}_o$	Translational offset
$\mathbf{q}$	Quaternion
$\mathbf{u}$,$\mathbf{y}$	Input vector, output vector
$\mathbf{A}$	State matrix
$\mathbf{B}$,$\mathbf{C}$	Input and output matrix
$G(s)$	Transfer function
n_0,d_0,d_1	Parameter transfer function
$\boldsymbol{\theta},\boldsymbol{\phi}$	Parameter vector, Measurement vector LSE
$\mathbf{z}$	Observation vector KF
$\mathbf{F}$,$\mathbf{G}$,$\mathbf{H}$	State matrix, input matrix, and observation matrix KF
$\mathbf{P}$	Process covariance matrix
$\mathbf{Q}$,$\mathbf{R}$	Process and Measurement covariance noise matrix
$\mathbf{W}$	KF Gain
$\mathbf{v}$	Residual observation KF
$\mathbf{S}$	Prediction covariance matrix KF
$\mathbf{r}$	Reference vector
$\mathbf{e}$	Error between x and y
v_x,v_y	Velocity in x and y-direction
h	Time steps total delay
τ	Time delay
δ_t	Time step
$\sum\mathbf{T}$	Total thrust rotors
$\mathbf{F}_z$	Gravity force
$\mathbf{v}_{tan}$	Tangential velocity path following
$\mathbf{v}_{approx}$	Approaching velocity towards the path

Abstract

Abstract

In the course of this thesis, a framework was developed to operate the Parrot Ar.Drone 2.0 autonomous in indoor environments. The quadcopter is controlled from a ground station computer. The data between both is transmitted wireless. Thus, inherent time delays occur. The proposed framework compensates the time delays and increases the overall performance of the closed loop. It consists of a observer-predictor algorithm based on a Kalman Filter and a PID velocity controller. Furthermore, a novel velocity field control concept is developed to control the quadcopter by using a velocity controller, instead of a position controller as it is norm in the literature. The velocity field control approach transforms the task of following a way-point defined path into a velocity field that points towards the path. Furthermore, obstacles are represented as potential field which then is derived to obtain a repulsive gradient field, too. In this work the theory of velocity field to perform the task of path following was successfully transformed from the field of ground robots to quadcopters.
All results were determined experimentally in an indoor environment.

Kurzfassung

Im Rahmen der Masterarbeit Entwurf einer Geschwindigkeitsregelung für einen fly-by-wireless Quadrocopter wurde ein System entwickelt, welches dazu beiträgt die Parrot Ar.Drone 2.0 autonom in Gebäuden zu betreiben. Das entworfene Regelungskonzept wird auf einem externen Computer ausgeführt. Die Kommunikation zwischen beiden Teilnehmern funktioniert über eine lokales Funknetzwerk (Wireless LAN). Dabei treten nicht deterministische Zeitverzögerungen sowie Paketausfälle auf, welche zu einer schlechten Gesamtdynamik des Systems führen. In dieser Arbeit wurde ein Beobachter-Prädiktor Algorithmus auf Basis eines Kalman Filters entwickelt, der die Zeitverzögerungen kompensiert. Dieser Algorithmus synchronisiert die vorhanden Beobachtungen um anschließend eine Prädiktion des Zustandsvektors durchzuführen, sodass es möglich ist, die vorhanden Zeitverzögerungen im System zu kompensieren. Darüber hinaus ist eine Anwendung für die Geschwindigkeitsregelung untersucht worden. Es wurde eine Pfadfolgeregelung basierend auf einem Geschwindigkeitsfeld implementiert. Das Geschwindigkeitsfeld ist ein Vektorfeld, so dass der Quadrocopter in Abhängigkeit seiner aktuellen Position einen Referenzvektor erhält, um dem Pfad zu folgen. Im nächsten Schritt wurde das System erweitert um automatisch Hindernissen auszuweichen. Ein Hindernis wird als Potentialfeld dargestellt und der Gradient dieses Feldes ist ein repulsives Geschwindigkeitsfeld. Im Rahmen dieser Arbeit konnte die Pfadfolgeregelung sowie das Ausweichen eines Hindernisses erfolgreich auf Quadrocopter angewendet werden. Alle Ergebnise wurden im Labor validiert.

1 Introduction

In the recent years, quadcopter gained the interest of researchers worldwide. A lot of progress has been achieved when it comes to autonomous outdoor flying tasks where an absolute position of the quadcopter is available due to GPS. Nevertheless, the research community still struggles with flying in indoor or GPS denied environments when no absolute position is available. Hence, this thesis aims to contribute to the current research in this field by developing a velocity controller for the Parrot Ar.Drone 2.0 to operate the quadcopter in an indoor environment. The Parrot Ar.Drone 2.0 is widely used within the research community. The specialty of the Ar.Drone is that it uses a wireless link to communicate with the ground station.

Quadcopters are already used in a wide range of tasks in the military and the civilian environment, such as: exploration, observation, object manipulation, and object transport. However, most of those applications are designed for outdoor environments. To perform such missions, both remote controlled and autonomous controlled UAVs are of interest. Furthermore, consumers buy quadcopters as a high-tech toy [4, 5].

A Quadcopter is a micro unmanned areal vehicle (UAV) and also named quadrotor helicopter. Quadcopters have similar dynamics as helicopters, while having a simpler mechanical structure. The rotors of the quadcopter have a fixed angle of attack. Moreover, quadcopters are highly nonlinear and inherent unstable systems [6]. By virtue of the developments in control theory, the stability of the quadcopter is ensured by adding artificial damping to the plant through feedback control. From the control perspective, quadcopters are an underactuated system that is force controlled. Force actuations are created through a change of the rotor speed and imply a rotational or translational motion to the quadcopter [7]. The internal controller allows the quadcopter to be extremely maneuverable and to hover on a spot. Different research groups proofed the capabilities of quadcopters to perform high dynamics maneuvers such as flips.

Although quadcopters gained popularity, professional quadcopter such as the AscTec Firely are too expensive for small research groups or for students to work with. The alternative to a non professional quadcopter would be a toy-quadcopter. The problem is, that most toy-quadcopters have a lack of sensors, are closed source, and are only controllable from a smartphone. The Parrot Ar.Drone announced in autumn 2010 overcame most of the issues. It was first designed for augmented reality games but attained soon attention in academia. The Parrot Ar.Drone has a variety of different sensors, a protection hull that makes it robust against crashes, and an internal controller which guarantees stability. Moreover, due to the use of an internal camera system, it got very good hover capabilities compared to other toy-quadcopters. The developer provides the researcher with a SDK (Software Development Kit) to control the quadcopter from an external computer over a wireless-link [8, 9]. Nowadays, the Ar.Drone and the successor the Ar.Drone 2.0 are widely used in research [1].

The Parrot Ar.Drone 2.0 has a variety of sensors such as Camera, IMU (Inertial Measurement Unit), and GPS (Global Positioning System). Those are required to perform tasks like autonomous navigation [9]. When flying in an outdoor environment usually GPS is used to estimate the absolute position of the quadcopter. However, when flying indoor no GPS signal is available. To avoid the lack of absolute position, research groups use motion capture systems in their laboratories. However, those systems are very expensive and the use case is limited to laboratories. Another approach is to use the internal camera of the quadcopter to estimate the absolute position. In conjunction with the position estimate a position controller is standard in research and application to perform tasks like navigation or path following. Therefore, a path is defined as a set of fixed way-points in the space [10]. The position controller, clas-

sical a PID control scheme, needs the position of the quadcopter and the way-points to calculate the control command.

1.1 Problem Statement

The goal of thesis is to develop a novel velocity controller based framework for the Parrot Ar.Drone 2.0 which will enable the quadcopter to operate autonomous tasks in an indoor environment with the advantage that no absolute position is required.

System overview

The quadcopter itself has an internal controller that guarantees stability and that controls the attitude dynamics. The velocity controller is executed from a ground station computer. Data between the ground station computer and the quadcopter is transmitted wireless. The system overview is displayed in figure 1.1 consisting of the wireless network, the quadcopter, and the ground station computer. Unfortunately,

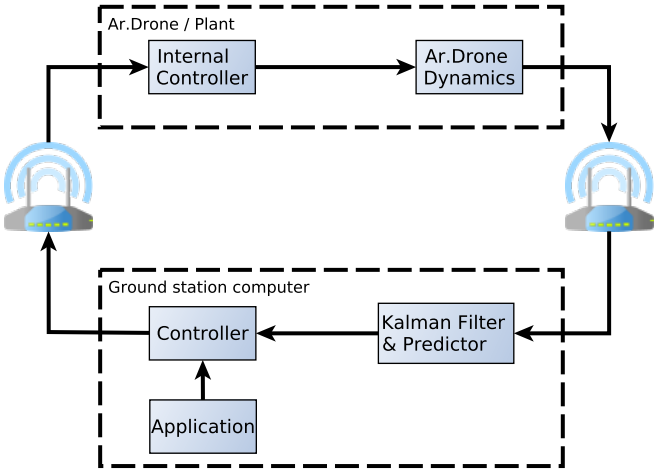


Figure 1.1: The figures shows the complete system consisting of a ground station computer and the Parrot Ar.Drone 2.0. Data between both is transmitted wireless.

the wireless network induces an inherent and non deterministic time delays to the closed loop. In control perspective this is equivalent to a system with a dead-time. This dead-time leads to a poor performance of the closed loop and induces oscillations into the system.

Application

The literature mainly focuses on a position controller. Position controller are widely for tasks such as path following. But a position controller is prone against disturbances. In addition, more complex tasks like obstacle avoidance require a dynamic recalculation intermediate positions and thus have high computational burdens.
To bypass those problems, and for many other applications a vector field based approach is preferred. Therefore, a velocity controller for the Parrot Ar.Drone 2.0 is developed. In conjunction with the velocity controller, a new approach is investigated, namely the velocity field control. This approach encodes the navigation task into a velocity field. It was already successfully implemented at ground robots and during the course of this thesis it is applied to the field of quadcopters [11, 12, 13].

Framework

The aim of this work is to develop a modular framework for the Parrot Ar.Drone 2.0. To do so, the following assumptions are made: the proposed algorithm should only use measurements available from the onboard sensors of the quadcopter. Furthermore, the monocular onboard camera should not be used

because depending on the application the video stream may not be available.
In particular, challenges of this work include: firstly, to identify the time delays of the closed loop, secondly, to handle the missing time stamps from the measurements of the quadcopter, thirdly, to deal with the sensor quality of the low-cost IMU and velocity measurement unit. To tackle all those problems we aim to develop a observer-predictor algorithm based on a Kalman Filter.
In the literature time delay compensation for the Parrot Ar.Drone 2.0 is already under investigation. All available solutions use the front facing camera of the Ar.Drone to perform parallel tracking and mapping (PTAM) [14, 15, 16]. They used a Kalman Filter in combination with PTAM. Those algorithms provide an absolute pose in space. However, the velocity controller does not require an absolute position estimate.

1.2 Outline

To achieve the mentioned goals this thesis is devided into 10 chapters. Starting with chapter 2:

1. **Chapter 2** focuses on the hardware and software which is used in the laboratory. It provides all the information for other research groups to build a similar setup. Furthermore, it explains the chosen wireless settings. Finally, the chapter briefly gives an overview about ROS.

2. In **chapter 3** the Parrot Ar.Drone 2.0 is introduced. The hardware and software specifications of the quadcopter are described in detail. This chapter summarizes all available information related to the internal controller of the Ar.Drone. Parrot, the company that developed the quadcopter, has not published any further details.

3. **Chapter 4** derives the dynamics of the quadcopter around the hover regime based on the equations of motions. This results in a linearized model. Afterwards, the linearized model is identified by performing experimental flights with the Parrot Ar.Drone 2.0.

4. **Chapter 5** is fourfold. First, it explains the features of time delayed systems. Then, it provides an overview about control concepts to handle and to compensate inherent time delays. After doing so, a concept for networked control systems, suitable for the setup in the laboratory, is introduced and based on that a discrete predictor scheme is derived from the Kalman Filter. Finally, a digital implementation of a PID control scheme with nonlinear modifications is explained.

5. In **chapter 6** the implementation of the proposed observer-predictor algorithm in a networked control system is explained. To do so, first the time delays of the system are analyzed. In the next step the measurements from the different sensors is synchronized and filtered. Finally, the filtered estimate is forwarded to the prediction scheme that carries out a prediction based on the last measurements, and the last computed control commands. Moreover, the nested control loop with an inner controller onboard of the quadcopter and an outer velocity controller on the ground station is explained.

6. In **chapter 7** an application for the in chapter 6 proposed framework is introduced. The task of path following is transformed by successively approaching a predefined way-points into a velocity field based control concept, that is able to counteract disturbances more effectively. Moreover, the new path following algorithm is expended to handle obstacle avoidance. To do so an obstacle is defined as a velocity field.

7. **Chapter 8** verifies the result of the prediction scheme and proofs that the time delay is successfully compensated. Furthermore, different test flights are performed and evaluated to show the capabilities, and limitations of the velocity controller. Finally, we point out that the quadcopter successfully performs the task of path following and obstacle avoidance.

8. In **chapter 9** and **chapter 10** the results of this project are summarized and the capabilities of the framework are highlighted. Finally, the last chapter proposes how to expand of the current setup and gives an outlook on further research.

2 Laboratory setup

In this chapter, we first describe the equipment in our laboratory. In section 2.1 we will provide all information necessary for other researchers to build an identical setup. Moreover, we will describe the environment in the laboratory, our indoor flying space and the limitations our laboratory. In section of this chapter 2.2 we provide an overview about ROS. In the last section 2.3 we present a list of the used standard units and the coordinate frame according to the ROS regulations.

2.1 Laboratory configuration

We shortly summarize the equipment used at the Roboterlabor at rmr TU Darmstadt. The Roboterlabor is a room of $5x5$m with a standard ceiling height of 2.8m. During the course of the thesis the following equipment was used:

1. **ground station:** the ground station was a powerful Linux computer with Ubuntu 14.04 which was used due to the long term support until 2019. It is important to note that we have not used a realtime kernel for Ubuntu to reduce the latency. In addition, the ground station itself had no wireless LAN module included, thus an external TP-LINK Archer T2UH WLAN-USB-Adapter was used. This WLAN stick is supported by Linux and has an antenna with a gain of 4dB to ensure an optimal communication between the drone and the ground station. The stick uses USB 3.0 and a frequency band of 2.4 gHz.

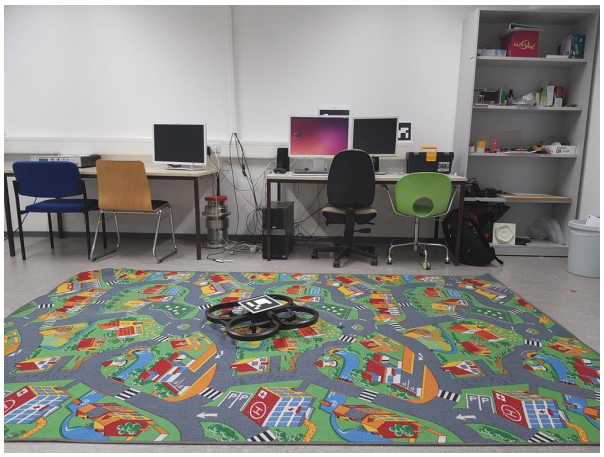
2. **Parrot Ar.Drone 2.0:** the laboratory consists of two Parrot Ar.Drone 2.0 [17, 1]. An old one, almost 1.5 years old, and a new one, bought for this project. We used both, since the parameters vary between the drones. The variation has two reasons: first during production small uncertainties arise in the components, and second parameters such as the rotor thrust or the rotor dynamics in general change over the duration. In particular crashes of the drone with objects damage the rotors and add scratches to them or bend them out of their plane. This influences the behavior of the drone during the flight. Hence, it is interesting to analyze if our proposed solution is able to deal with this uncertainties between the different drones.

3. **flying arena:** most office buildings and universities as well as our laboratory have solid floors with an unity gray color. Such floors are characterized by a lack of features. To ensure an accurate velocity estimation of the quadcopter, the ground has to be featured, because the onboard velocity measurement uses optical flow and corner detection. Details to the velocity measurement are in chapter 3. Therefore, the problem is counteracted by a children play carpet which features a variety of items. The carpet is used as ground to perform the experiments on it. The size of the carpet is $2x3$m and it mainly limits the flying space.

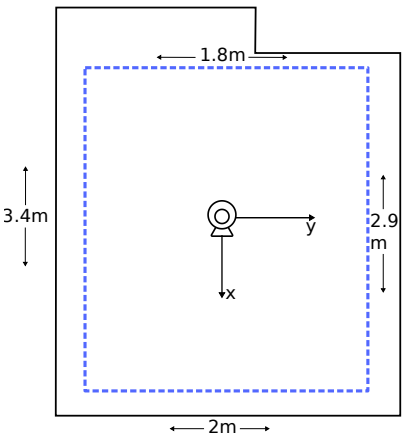
4. **camera system:** the laboratory has a Logitech HD C615 fixed on the ceiling looking downwards. The camera has a resolution of $1920x1080$pixels at a maximal frame rate of 30 frames per second, a wide angle objective, and is able to overview the flying space. The camera allows the user to manually adjust the exposure time and to set the focus level. The camera is only used for validation and it is used in the application proposed in chapter 7.

For the communication between the Parrot Ar.Drone 2.0 and the ground station a wireless LAN is set up which uses the channel 6 at the 2.4 gHz network as default. In our laboratory a lot of jitter disturbs

(a) Front



(b) Side



(c) Layout laboratory

(d) Field of view ceiling camera

Figure 2.1: The pictures show the experimental setup. The flying space of the Ar.Drone is the high textured carpet. The background of picture (b) shows the ground station computer equipped with Ubuntu. Figure (c) shows the layout of the laboratory. The blue line marks the flying arena. The camera is in the center of the carpet. Figure (d) shows the field of view of the ceiling camera with full HD resolution (1980x1920 pixel).

at channel 6 which is caused by the eduroam network (wireless network of our university) so that we had the most stable connection at channel 4. We wrote a script and stored it on the drone to switch the channel. However, the drone did not store the channel thus after every restart of the system the script has to be run again. All methods were implemented using C++ and Python in combination with ROS. So we give a brief introduction to ROS in the next section.

2.2 ROS

ROS (robot operating system) is an open source software-framework mainly supported by the open source robotics foundation (osrf) [18]. Nowadays, ROS is the standard tool in robotic research with a great community of researchers who are willing to discuss and publish their packages to extend the features of ROS. In robotics low latencies are crucial to control an unstable system, ROS provides this but is not a real time operating system.
The framework of ROS is designed to be distributed and modular. This allows the user to run several

processes in simultaneously. In ROS processes are called nodes and are loosely coupled by using the ROS communication framework. ROS consists of three core components:

1. **Communication infrastructure:** ROS allows a synchronous communication over services, or an asynchronous data transfer over topics. Topics are mostly used to pass messages between nodes. Furthermore, it allows to record and playback all messages and thus provides a tool for off-line analysis.

2. **Robot-specific-features:** ROS provides standard messages for a variety of different data such as measurements from sensors like IMU, pose of a robot, and velocity (linear and angular). Moreover, it got built in tools to perform coordinate frame transformations.

3. **Tools:** ROS provides the researcher with different tools. The most famous ones are Rviz and rqt. Rviz visualizes the data of sensors and the pose of the robot in the three-dimensional space. rqt is an interface to visualize data. For instance rqt plots the data of sensors in a graph, displays how the nodes interact with each other, or summarizes the information of each message in a table.

Distro	Release date	EOL data
Kinetic Kame	2016-05-23	2021-05-30
Jade	2015-05-23	2017-05-30
Indigo	2014-07-22	2019-04-30
Hydro	2013-09-04	2014-05-31
Groovy Galapagos	2012-12-31	2014-07-31

Table 2.1: Version History of ROS

Table 2.1 shows the different versions of ROS. We use the Indigo distribution due to the long term support (end of life 2019-04-30).

2.3 Standard Units of Measure and Coordinate Conventions

In this work all data between the nodes is transmitted by using the ROS REP-103 conventions [19]. This provides a simple and clarified way to communicate. However, within an algorithm different conventions may be used.
The used units are shown in 2.2. The conventions for the coordinate frame are as following:

Unit	Quantity
time	second
length	meter
mass	kilogram
angle	radian
frequency	hertz

Table 2.2: Measurements Units ROS DEP-103

- x-direction (pointing forward)
- y-direction (pointing left)
- z-direction (pointing up)

3 Hardware: Quadcopter

3.1 Parrot Ar.Drone 2

The Parrot Ar.Drone was introduced in 2010 and was developed with the aim to be an easy to use drone for the mass market. First the drone gained attention as high-tech toy for virtual reality applications. However, soon it gained the interest of universities and research institutions as result of its low price under 300€, good hover capabilities, and the robustness against crashes. Hence, nowadays the drone is widely used in the field of robotics and computer vision.
In the first section the hardware of the parrot Ar.Drone 2 and focus on the internal sensors are described. Then communication protocols between the ground station and the quadcopter are explained. The used protocols are independent of the used hardware. In the last section the onboard controller are discussed in detail. The quadcopter itself, in open loop configuration is an unstable system. Therefore, it is equipped with an advanced embedded control system to guarantee stability of the system.
In this thesis Ar.Drone will be used as acronym for the Parrot Ar.Drone 2.0.

3.1.1 Hardware

The Parrot Ar.Drone 2 has four rotors fixed on a carbon-skeleton in x-configuration (see figure 3.1). The Ar.Drone is delivered with an indoor hull which protects the propellers in the case of a crash with a wall, furniture or other obstacles and a light outdoor hull but without protection for the propellers. As a result of the very protective polypropylene hull the drone has survived minor and heavy crashes during the numerous flights of this project. The disadvantage of the indoor hull is that it decelerates the system dynamics and increases the sensitivity against wind gusts. Therefore, if higher flying speeds or a very dynamic flight maneuver are required, the outdoor hull should be used. The drone weights with the indoor hull $420g$, and with the outdoor hull $380g$. A battery charge lasts for up to $15min$ flight time depending on the capacity density of the battery [1, 20].

Figure 3.1: This image shows the indoor hull (left) and outdoor hull (right) as well as the carbon fiber x-skeleton on which the four rotors are fixed.[1]

If the battery is connected to the drone it automatically boots and sets up an ad-hoc wireless local area network. It is using the $2.4GHz$ wireless LAN band and is limiting the output power to $100mW$ due to EU regulations. This network is used for data transfer with the Ar.Drone and limits the range of connectivity to less than $50m$. The Ar.Drone has a 1-GHz-32-bit ARM processor plus a 800-Mhz processor for video processing from Texas Instruments and is running a BusyBox GNU/linux distribution [8].

3.1.2 Sensor equipment and camera system

The Parrot Ar.Drone 2 is equipped with numerous sensors. The camera system consists of a main camera in the front, looking ahead, and a second camera at the bottom, looking down. Moreover, it is equipped with a accurate Gyroscope to measure the tilt angles and a magnetometer for orientation. The altitude is measured based on a pressure and a ultrasound sensor.

Camera system

The features of both cameras are displayed in table 3.1. The main camera (front) is designed for flying the drone with a smartphone or tablet and has HD resolution. Furthermore, the forward facing camera may produce blur images due to rapid drone movements such as translation from hovering to movement.

Camera	Front	Down
Resolution	720p	320p
Frame rate	30fps	60fps
Field of view	92°	64°

Table 3.1: Data sheet Parrot AR.Drone 2 cameras

The downward facing camera has a bad image quality and is used to estimate the velocity. In the case of low velocity and a highly textured ground the velocity is estimated based on corner detection. For higher speeds or a solid ground with low contrast scenes optical-flow technique is used for a velocity estimation. Corner detection reaches a better accuracy than optical-flow technique. Hence, it is used for hovering [2, 1, 17].

Figure 3.2: The front of the drone with the centralized HD is displayed.[1]

Shadows, created by the lightning of the ceiling, follow the movement of the quadcopter and are recognized by the corner detection or optical-flow algorithm as an additional movement of the quadcopter. The internal controller of the Ar.Drone tries to counteract this non-existent movement which may lead

to oscillations and therefore to an unstable system behavior [21, 22].
The computational power of the drone and the bandwidth are limited. Therefore, only the video stream of one camera can be linked to the user or ground computer [14].

Inertial Measurement Unit and Magnetometer

The inertial measurement unit (IMU) consists of a gyroscopes and accelerometers and costs less than 10€. The data sheet of the IMU is shown in the table below:

Sensor	Name	Axis
Gyroscope	Epson XV3700	1-axis (z)
Gyroscope	Invensense IDG500	2-axis (x and y)
Accelerometer	Bosch BMA 150	3-axis
Magnetometer	TBA	3-axis

Table 3.2: Data sheet IMU Parrot Ar.Drone 2

The accelerometer is a 3-axis Bosch BMA 150. It has a +/- 2g range and a +/- 50mg precision. The gyroscope is a 2-axis (x and y axis) Invensense IDG500 analog sensor and measure rates up to 2000°/$second$. For the vertical-axis an Epson XV3700 is used. This gyroscope has an auto-zero function to minimize the drift over time. The IMU is running at a rate of 200Hz. The Ar.Drone has in addition to the IMU a 3-axis Magnetometer with a precision of +/- 6°.
Experiments showed that the gyroscope measures the yaw angle with the error 4° per minute. The gyroscope measures the pitch and roll angle with the error less than 0.5° per minute. On the basis of the high accuracy of the pitch and roll angle both measurement were used as reference value. The yaw angle measurement was always exposed to huge drifts over the flight time [1, 20].

Altimeter

The altimeter consists of an ultrasonic sensor for low altitudes and a pressure sensor for higher altitudes. The first sensor is able detect vertical displacements as large as 6m at a rate of 25Hz. The pressure sensor allows the drone to measure its altitude at any height with the error +/- 0.5m. The experiment showed that the ultrasonic sensor is not working correctly on unhitched ground as grass hayfields. Furthermore, the altitude measurement showed a drift in height during flights over solid ground. The internal controller of the Ar.Drone will counteract this height differences. This results in small vertical displacements.

Additional sensors

The Parrot Ar.Drone 2 includes a mini USB connector to plug in additional hardware [17].

- Flight recorder (GPS)
- laser beamer

3.1.3 Software

The Parrot Ar.Drone 2 can be controlled via an iPhone or android smartphone, as well as with a tablet computer or a personal computer with any operating system. The system is mainly designed to be controlled with a smartphone or tablet. In particular the application for the iPhone allows changing a numerous amount of settings such as trimming or updating the software of the four propellers of the quadcopter. In addition a software development kit (SDK), written in plain C code, is available for the

use with Linux. It is highly recommended to use the SDK on a Linux computer, since on Windows heavy bugs are present.
A huge limitation of the Parrot Ar.Drone 2 is that neither the onboard software is accessible nor any documentation is available. It is still possible to modify the software via a telnet command line but this approach can lead to irrecoverable damages without knowledge about the system. Thus, the predefined interfaces were used for communication [1]. The software is separated into three communication channels:

- **Navigation:** This channel transmits data related to the state of the drone and the odometry data.
- **Video:** transmits the video stream of one of the two cameras to the user/ground PC. It also handles the selection process.
- **Command:** this channel sends control commands to the drone. Commands like take off or landing are predefined procedures. Commands can also be sent manually as reference value for the orientation of the quadcopter.

3.1.4 Control Architecture

The embedded control architecture consists of a data fusion algorithm and a nested control system. The control is running at the same rate as data acquisition, namely at a rate of $200Hz$. The system has a human in the loop [8].
The main goal of the Ar.Drone is to be an easy to fly quadcopter for the mass market. Therefore, there is no possibility to control the rotor thrust or rotor speed directly. Instead the user can send a command for the desired tilt angle of the quadcopter. This idea is based on a human controlling the drone with a smartphone. This means a bending angle of the phone is directly converted into a desired tilt angle for the drone. The system inputs of the Parrot Ar.Drone 2.0 are as follows:

- Pitch angle θ (y-axis)
- Roll angle ϕ (x-axis)
- Yaw angle rate $\dot{\psi}$ (z-axis)
- height rate $\dot{z}$ (z-axis)

The height and yaw angle are introduced as rate since both measurements have huge drifts over time.
For autonomous flying applications the human is replaced by a high task mission planner. The architecture of the control system is built as a finite state machine with various modes such as: hovering, take off, landing and forward flight. If the Ar.Drone switches from the forward flight mode to the hovering mode the Ar.Drone follows an off line calculated trajectory to reach the steady state where the velocities and tilt angles are zero and the altitude is held constant. For the modes take off and landing control procedures are implemented which do the take off or landing fully autonomous without the involvement of a human. Those tasks are complex and would otherwise take users several hours of training to perform them safely. If the reset command is triggered during flight it will cause a crash since the propellers will instantly stop spinning.
The main focus of this thesis is set on the flying forward mode in which the attitude is controlled by the embedded controller. The control is realized by two nested loops. The outer loop is the attitude control loop and computes an angular rate set point. The control input is the difference between the desired tilt angle and the estimated tilt angle based on the data fusion algorithm. The control is done with a proportional integral (PI) control. The inner loop tracks the angular rate and controls the motors with a proportional (P) controller [2].
This low level control system guarantees stability of the drone and is user friendly. The complexity of

the automatic control is not visible for the user. Thus, the platform is suitable to build more advanced control and guidance algorithms.
Parrot has not published any further information related to the control system. Hence, no parameters of the controllers are available.

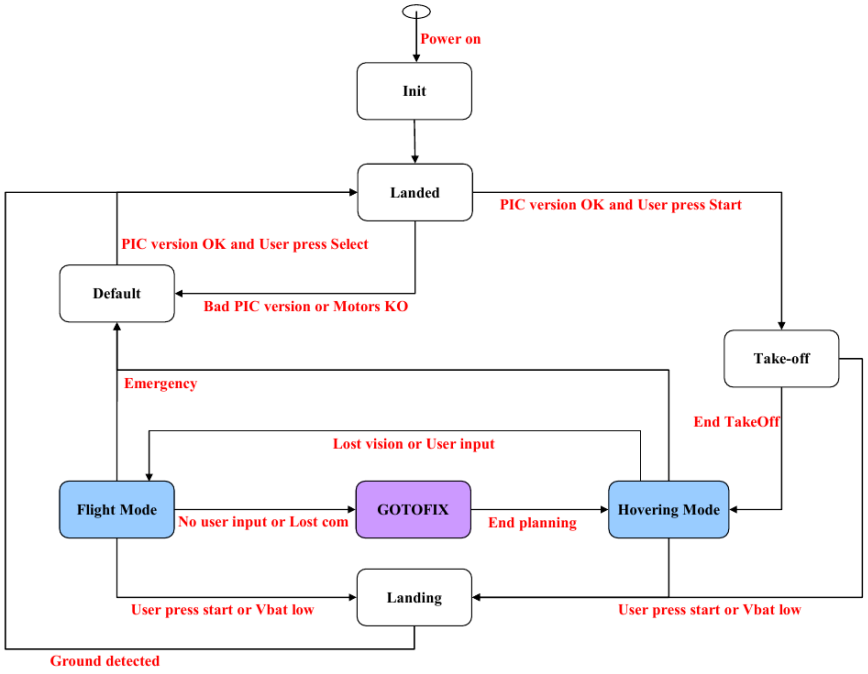


Figure 3.3: State machine with different modes of Parrot Ar.Drone 2 [2].

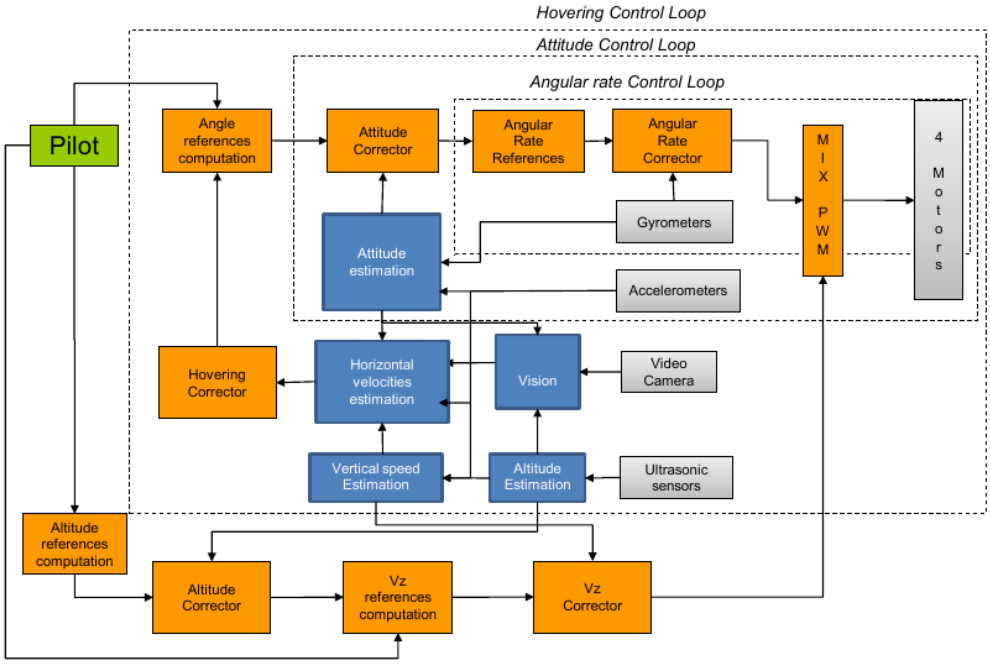


Figure 3.4: Data fusion and control architecture [2].

4 Modeling and Identification of Parrot Ar.Drone 2

In this chapter, the dynamics of a quadcopter are derived around the hover regime from the equations of motion. Based on this information as well as the information about the quadcopter from chapter 3 a simplified model of the quadcopter is created. Afterwards, a system identification of the Parrot Ar.Drone 2 is performed to obtain estimates for the key parameters of the drone.

4.1 Dynamics

The principle of operation of a quadcopter consists in the generation of a net force through spinning of the four rotors. Quadcopter are characterized by their set of four identical rotors. Through varying the speed of each rotor separately the rotational and translational dynamics of the drone are controlled.

- **Hover:** describes a steady state behavior of the quadcopter where all velocities are zero and the height is hold constant. In this state all four rotors spin with the same revolution speed. The Parrot Ar.Drone 2 has a built in (nonlinear) hover controller which first transfers the system from a dynamic mode (like forward flight) into hover state and second steadies the pose of the quadcopter in space.
- **Tilt of the quadcopter:** to induce lateral translational dynamics a tilt of the quadcopter is required. As example the acceleration in x-direction is generated through a rotation around the y-axis and is referred to as pitch angle θ. Thus, the pitch angle can be interpreted equivalent to a force acting at the center of the quadcopter. The tilt angle is induced by increasing the rate of rotors {3,4} and decreasing the rate of rotors {1,2}. In conclusion, the pitch and roll angle are adjusted by applying more or less thrust to the different set of rotors. The sets are: to induce a pitch angle {1,2} and {3,4}, and to induce a roll angle {1,3} and {2,4}.
- **Rotation around the z-axis:** the rotors produce thrust as well as torque in the center of each rotor. The rotors are set up in counter rotating pairs, such that the torque cancel each other in the center of the quadrotor, provided that they are all rotating with the same speed. A rotation around the z-axis of the quadcopter is generated through a imbalance of momentum in the center of mass. The imbalance is induced by applying more or less thrust to the rotor pairs {1,4} and {2,3}

The rotational speed of each rotor is an actuator of the system. Furthermore, the system has only four actuators for six degrees of freedom. This means the quadcopter is an underactuated system. In this case it is still controllable but it is limited in the choice of trajectories to go from an initial point to a fixed point in space in a finite amount of time [1, 6, 23].

4.1.1 Preliminaries

Coordinate Frames

The dynamics are expressed in terms of the body-frame, such are the velocities and orientation. The origin of the coordinate frame is defined through the center of mass. In detail, the middle of the front camera is aligned with the x-axis and points forwards. The y-direction points to the right and z-direction

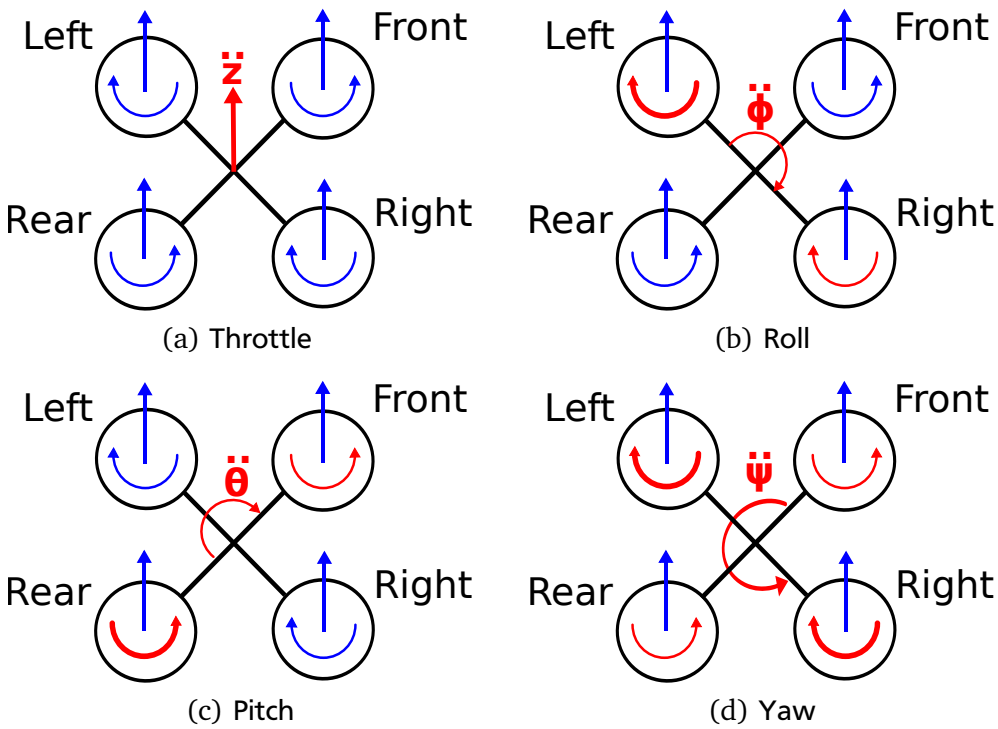


Figure 4.1: Drone maneuvers are obtained by changing the pitch, roll and yaw angle of the quadcopter. In Figure (a) all four rotors are spinning with the same rotational speed. This maneuver is referred to as hover $\ddot{z} = 0$. Vertical dynamics are induced by an increase or decrease of the rotational speed of all four rotors. A rotation around the y-axis is induced in figure (b) by increasing the rotor rate of the left rotor and decreasing the rotor rate of the right rotor. Figure (c) shows a similar maneuver for the x-axis. The rotation around the z-axis (figure (c)) is caused by a pairwise increase of the left and right rotor speed and a pairwise decrease of the front and rear rotor speed.

points upwards (4.2). The body-frame is designed to express the dynamics of the quadcopter. It is not suited to provide a global position. Hence, a world-frame is used to represent the position of the quadcopter in the space. The world-frame is created according to REP-103 convention to ensure a consistent use of the coordinate frames [19]. The world-frame follows the same conventions as the body-frame, with one exception, the y-direction points to the left, instead of the right direction. Both frames use the right-hand rule as convention for the orientation:

1. roll angle ϕ is defined as rotation around the x-axis.
2. pitch angle θ is defined as rotation around the y-axis.
3. yaw angle ψ is defined as rotation around the z-axis

The system has six degrees of freedom, three translational degrees (x-,y- and z-axis) and three rotational (ϕ, θ, ψ) degrees of freedom [6].

Rotation in the reference frame

The transformation between the body-frame and the world-frame is defined through a $3x3$ rotation matrix and a $3x1$ translation vector. The rotation matrix $\mathbf{R}(\phi,\theta,\psi)$ consists of three independent rotations. The translation vector $\mathbf{x}_o$ describes an offset in the x-, y- and z-direction between the origins of a

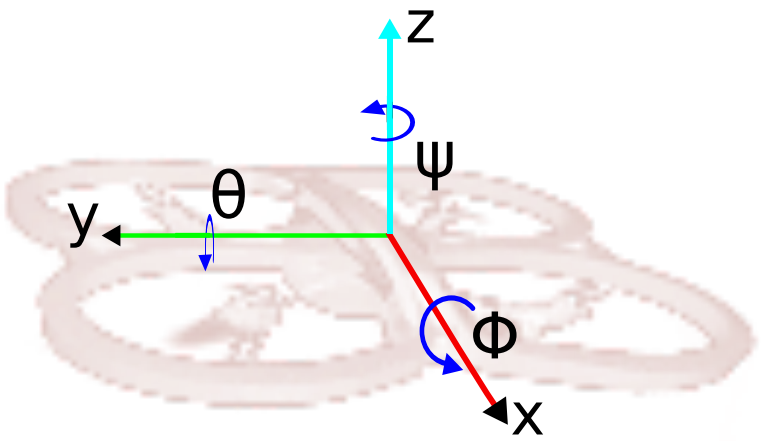


Figure 4.2: Body-frame of the quadcopter.

movable reference frame and a fixed world-frame. Hence, a translation offset is expressed through $\mathbf{x}_o$ in world coordinates.

$$\mathbf{R}(\phi,\theta,\psi) := \begin{pmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\psi \\ -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & \sin\phi\cos\theta \\ \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi & \cos\phi\cos\theta \end{pmatrix} \tag{4.1}$$

The rotation between both frames is given through $\mathbf{R}(\phi,\theta,\psi) = \mathbf{R}(\phi)\mathbf{R}(\theta)\mathbf{R}(\psi)$. Firstly, the world-frame is rotated around the x-axis (ϕ Rotation). The rotation does not change the x-coordinate of the point $\mathbf{x}_b$. Secondly, the world-frame is rotated around the y-axis (θ Rotation) and last it is rotated around the z-axis (ψ Rotation).

$$\mathbf{x} = \mathbf{x}_o + \mathbf{R}(\phi,\theta,\psi)^{\mathrm{T}}\mathbf{x}_b \tag{4.2}$$

Equation 4.1 describes the transformation of a pose consisting of a position and orientation from between to reference frames by using the rotation matrix. The transformation depends on the order it is executed and may vary through the different coordinate frames [24].

Quaternion

The order of Rotation between the axis has a crucial role and the transformation process and is not uniformly defined. Moreover, in total 24 different definitions of coordinate frames based on euler angles exist. Both reasons lead to errors when using the euler representation. Therefore, quaternion are used to represent the orientation. Quaternion, as the name suggests, is a 4-tulpe. It is written as: $\mathbf{q} = (q_0,q_1,q_2,q_3)$ and the sum of all elements is always equal to one. In addition, quaternions have no singularities and are widely used in the field of aviation.

4.2 Modeling of the Parrot Ar.Drone 2.0

In this section, the non linear dynamics of the drone are simplified for an operating regime around the hover point.

4.2.1 Quadrotor model near hover operation

Around the hover regime only the basic principles of motion are used, instead of complex aerodynamic effects in the case of fast dynamics. A operating point next to the hover regime is shown in figure 4.3.

The rigid quadcopter model is subject to two forces: the gravity of the quadcopter $\mathbf{F} = mg$ and the thrust from each rotor in total $\sum \mathbf{T}$, that point in the opposite direction as the gravity force. Moreover, each rotor creates a torque. Both (torque and thrust) are proportional to the rotor speed. For the case of this work the translational dynamics (velocities in x- and y-direction) are of primary interest. Secondly, the attitude dynamics are of interest, since a change in θ and ϕ serve to induce translational acceleration to change the velocity and position of the system. For the sake of clarity the attitude dynamics and the translational dynamics are coupled but the attitude dynamics between the x-axis and y-axis are decoupled. The yaw and height dynamics are decoupled from the other dynamics.
For our purpose we take no further look in the dynamics of the quadcopter, since the Parrot Ar.Drone 2.0 is equipped with a closed-loop attitude regulation. Hence, the Ar.Drone is modeled with a internal controller which guarantees the closed loop stability of the system [2, 9]. The available system inputs are as follows:

$$\mathbf{u} := \begin{pmatrix} \phi_{ref} \\ \theta_{ref} \\ \dot{\psi}_{ref} \\ \dot{z}_{ref} \end{pmatrix} \tag{4.3}$$

The input vector $\mathbf{u}$ contains the reference values for: the roll angle, the pitch angle, the yaw rate, and height rate.

4.2.2 Attitude dynamics

We assume the attitude dynamics are modeled as a simplified second order model. Furthermore, the system is stable and the pitch and roll dynamics are identical to one another. The attitude dynamics between x- and y-axis are decoupled and thus each axis is represented as a SISO system.

$$G_\theta(s) = \frac{\theta}{\theta_{ref}} = \frac{n_0}{s^2 d_1 + s d_1 + d_0} \qquad G_\phi(s) = \frac{\phi}{\phi_{ref}} = \frac{n_0}{s^2 d_1 + s d_1 + d_0} \tag{4.4}$$

4.2.3 Lateral translational dynamics

The quadcopter is modeled as a point of mass model. Usually the simplified translational dynamics are based on the equations of motion in an equilibrium state. This section points out the errors in the traditional assumption and introduces a new revisited model of the quadcopters translational dynamics that achieves more accurate results.

Usual interpretation

To induce a translational acceleration the quadcopter is tilted sideways. The operating point is shown in figure 4.3. In this state the thrust vector $\sum \mathbf{T}$ consists of a vertical component $\mathbf{F}_z$ in the opposite direction of the gravity and a horizontal component $\mathbf{F}_x$ to induce a movement in the x-y plane.
The tilt angle θ induces a sideway tilt out of the x-y plane. The same angle is also spanned between the vectors $\mathbf{F}_z$ and $\sum \mathbf{T}$. The tilt angle splits the total thrust $\sum \mathbf{T}$ that acts on the quadcopter into a vertical and horizontal component. By increasing the tilt angle the horizontal force increases and vice versa. However, the total thrust $\sum \mathbf{T}$ is constant. We assume the vertical dynamic is in a equilibrium state in which the vertical force $\mathbf{F}_z = \sum \mathbf{T} \cos\theta$ is equal to the gravity force: $\sum \mathbf{T} \cos\theta = mg$. This leads to the horizontal force [23, 9]:

$$\begin{aligned} \mathbf{F}_x &= mg \frac{\sin\theta}{\cos\theta} \\ m\ddot{x} &= g \tan\theta \end{aligned} \tag{4.5}$$

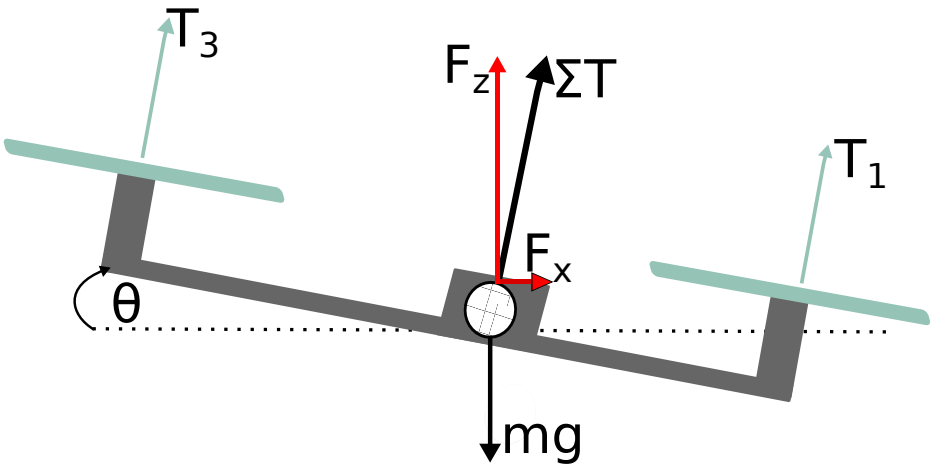


Figure 4.3: The tilt angle θ induces a sideway movement of the quadcopter under the assumption of operation near the hovering regime.

By virtue of the operation regime near the hover point the small angle assumption is used such that: $\tan\theta \approx \theta$ for small θ. This leads to:

$$a_x = g\theta \tag{4.6}$$

The same approach is used to model the velocity in the y-direction. Just to conclude, the assumptions based on the vertical equilibrium state neglect the coupling of the x-y axis. The air drag depending on the second order velocity is neglected due to the low speeds around the hover regime. This leads to the transfer function of the translational dynamic:

$$G_{v,x} = \frac{v_x}{\theta} = \frac{g}{s} \qquad G_{v,y} = \frac{v_y}{\phi} = \frac{g}{s} \tag{4.7}$$

Revisited interpretation

The revisited interpretation is based on non-linear rotor dynamics. The equation of motions are derived in [25]. The lateral translational dynamics are modeled as follows:

$$m\mathbf{V} = m\mathbf{g} + k_T \sum_{i=1}^{4} \omega_i^2 \mathbf{z} - \lambda_1 \sum_{i=1}^{4} \omega_i \tilde{\mathbf{V}} \tag{4.8}$$

$\mathbf{V}$ = velocity body frame $\quad g$ = gravity vector
m = mass of the quadcopter $\quad \tilde{\mathbf{V}}$ = projection of the propeller plane (4.9)
ω_i = rotational velocity of each rotor $\quad \lambda_1$ = positive constant rotor drag coefficient

The initial situation is the same as in the usual interpretation. The quadcopter is tilted sideways. In the moment an arbitrary small tilt angle is present the quadcopter starts to move sidewards.
Equation 4.8 shows two key aspects of the revisited interpretation. Firstly, the thrust vector (second term) points in the z-direction and therefore has no influence related to the sidewards movements of the quadcopter. Again, the equilibrium state for the vertical dynamics is assumed such that the height is held constant. Secondly, last term is proportional to the velocity of the quadcopter. We suppose the propeller consists of two blades, an advancing and a retreating blade. The advancing blade experiences a higher velocity with respect to free air than the retreating blade due to the non zero velocity of the whole quadcopter. This causes an imbalance that creates a force to flip the blades up and down as they rotate. Thereby, the blades are bended and rotate out of their plane. This causes a thrust force to tilt the quadcopter in the opposite direction of movement. The behavior is called blade flapping and modeled

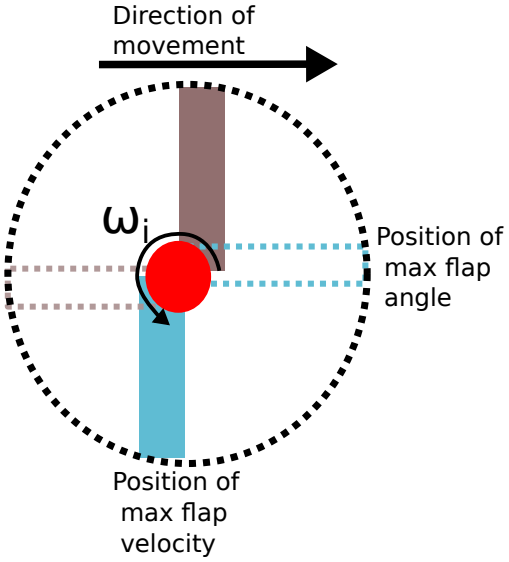


Figure 4.4: The rotor consists of two blades. The rotor blades rotate out of their plane due to a bending of the blade caused by an effect called blade flapping. The flap angle continuously changes during one rotation. After a half spin the advancing and retreating blade switch.

by the last term of equation 4.8 [25, 26].
The equation 4.8 is simplified as follows:

$$\begin{aligned} \dot{v}_x &= g \sin\theta - \frac{k_1}{m} v_x \\ \dot{v}_y &= -g \cos\theta \sin\phi - \frac{k_1}{m} v_y \\ k_1 &= \lambda_1 \sum_{i=1}^{4} \omega_i \end{aligned} \tag{4.10}$$

The parameter k_1 is a positive constant. It is constant because the rotating speed of the rotors is fairly constant during a smooth flight. Furthermore, small angle assumptions are used such that: $\cos\theta = 1$, $\sin\theta = \theta$ and $\sin\phi = 0$ in equation 4.10. The linearized model is as follows:

$$\begin{aligned} a_x &= g\theta - \frac{k_1}{m} v_x \\ a_y &= 0 \end{aligned} \tag{4.11}$$

This leads to the transfer function for the revisited model:

$$G_{v,x} = \frac{v_x}{\theta} = \frac{g}{s + \frac{k_1}{m}} \tag{4.12}$$

4.2.4 Vertical translational dynamics and Yaw dynamics

The vertical translational dynamics behave mostly linear and are reasonable modeled as a first order system. In the sake of clearance, the input of the system is the height rate due to the simplicity that a zero input leads to a constant height, no change in the state variables of the system. The same is valid for the yaw angle dynamics. Both dynamics are modeled as follows:

$$G(s) = \frac{z}{\dot{z}_{ref}} = \frac{n_{z,0}}{s d_{z,1} + d_{z,0}} \qquad G(s) = \frac{\psi}{\dot{\psi}_{ref}} = \frac{n_{\psi,0}}{s d_{\psi,1} + d_{\psi,0}} \tag{4.13}$$

4.2.5 State space model of the quadcopter

The combined state space model of the whole quadcopter consists of 12 states. The **A** matrix indicates that movements between the x-y axis are decoupled. In particular, it indicates the coupling between the lateral translational velocities and the tilt angles.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \qquad \mathbf{y} = \mathbf{c}\mathbf{x} \tag{4.14}$$

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \\ v_x \\ v_y \\ v_z \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} \quad \mathbf{u} = \begin{pmatrix} \phi_{ref} \\ \theta_{ref} \\ \dot{\psi}_{ref} \\ \dot{z}_{ref} \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} z \\ \phi \\ \theta \\ \psi \\ v_x \\ v_y \end{pmatrix} \quad \mathbf{C} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & g & 0 & \frac{-k_1}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & g & 0 & 0 & 0 & \frac{-k_1}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & -d_{z0} & 0 & 0 & 0 & 0 & 0 & -d_{z1} & 0 & 0 & 0 \\ 0 & 0 & 0 & -d_0 & 0 & 0 & 0 & 0 & 0 & -d_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -d_0 & 0 & 0 & 0 & 0 & 0 & -d_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -d_{y0} & 0 & 0 & 0 & 0 & 0 & -d_{y1} \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{z0} \\ n_0 & 0 & 0 & 0 \\ 0 & n_0 & 0 & 0 \\ 0 & 0 & n_{y0} & 0 \end{pmatrix} \tag{4.15}$$

The measured outputs of the state space model are: the tilt angles (ψ,ϕ,θ), the velocities in the x-y plane (v_x,v_y) and the height(z).

4.3 Identification

In this section, the quadcopter dependent quantities that appear in the dynamic models, described in section 4.2, are identified. The data were recorded in real-time mode with the maximum update rate of 200Hz. The actuating variables are defined as fraction of the maximum allowed input. Hence, they have values in the range of $\epsilon[-1,1]$. The procedure for parameter identification is as follows [20, 26]:

Property	Maximum value
θ	12°
ϕ	12°
$\dot{\psi}$	$100° \frac{1}{s}$
$\dot{z}$	$0.7 \frac{m}{s}$

Table 4.1: List of maximum input values for the indoor setup of the Parrot Ar.Drone 2.0 [27].

- The transfer function was identified without considering the dead-time of the system. To do so, the step input was manually aligned with the step response in time.
- The measurements of the quadcopter are overlain by noise. To reduce the influence of noise, a third order smith low-pass filter was used.
- The experimental data are split into data for identification and validation.
- Black-box model: the identification is based on assumptions from the modeling section. Thus, only the parameter of the transfer function are identified.
- The identification process is carried out with system identification toolbox of MATLAB. In addition, we manually used the least square estimation.

Transfer function

A linear model can be stated as transfer function:

$$G(s) = \frac{s^m b_m + s^{m-1} b_{m-1} + \cdots + s b_1 + b_0}{s^n a_n + s^{n-1} a_{n-1} + \cdots + s a_1 + 1} = \frac{\mathrm{B}(s)}{\mathrm{A}(s)} \tag{4.16}$$

For n a positive integer B(s) and A(s) are polynomials and the order of A(s) is higher than the order of B(s). The task is to identify the parameters a_i and b_i from the input-output measurements. A least square estimation (LSE) is used to obtain the parameters [28, 25]. The LSE requires that the transfer function equals the structure of equation 4.16. Furthermore, the parameter vector $\boldsymbol{\theta}$ and the measurement vector $\boldsymbol{\psi}$ are defined:

$$\begin{aligned} \boldsymbol{\theta} &= [a_1\ a_2\ \ldots\ a_n \quad | \quad b_0\ b_1\ \ldots\ b_m]^T \\ \boldsymbol{\psi}^T &= [-y(t)^1\ \ldots\ -y(t)^n \quad | \quad u(t)\ \ldots\ u(t)^m] \end{aligned} \tag{4.17}$$

Afterwards, the equations are discretized with the sample rate δ. The aim of the LSE is to minimize the quadratic error in $\mathbf{y} = \boldsymbol{\psi}\boldsymbol{\theta} + \mathbf{e}$. The optimal result is reached if $\mathbf{e}$ has its minimum value. This leads to equation:

$$\hat{\boldsymbol{\theta}} = (\boldsymbol{\psi}^T \boldsymbol{\psi})^{-1} \boldsymbol{\psi}^T \mathbf{y} \tag{4.18}$$

The estimated parameters are $\hat{\boldsymbol{\theta}}$. The LSE is used in the identification of the attitude dynamics in equation 4.20 and in the identification process of the translational velocities in equation 4.23.

Identification of attitude dynamics

The pitch and roll dynamics were identified by using a step input and a manual flight with the aim of encouraging wider spectrum of frequencies in the attitude dynamics. First, we present the results for the case of the step input. The actuating variable was set in a range of u $\epsilon[-0.2, 0.2]$. Bigger reference values for the pitch and roll angle were not applied due to the limitations of space in the indoor environment.

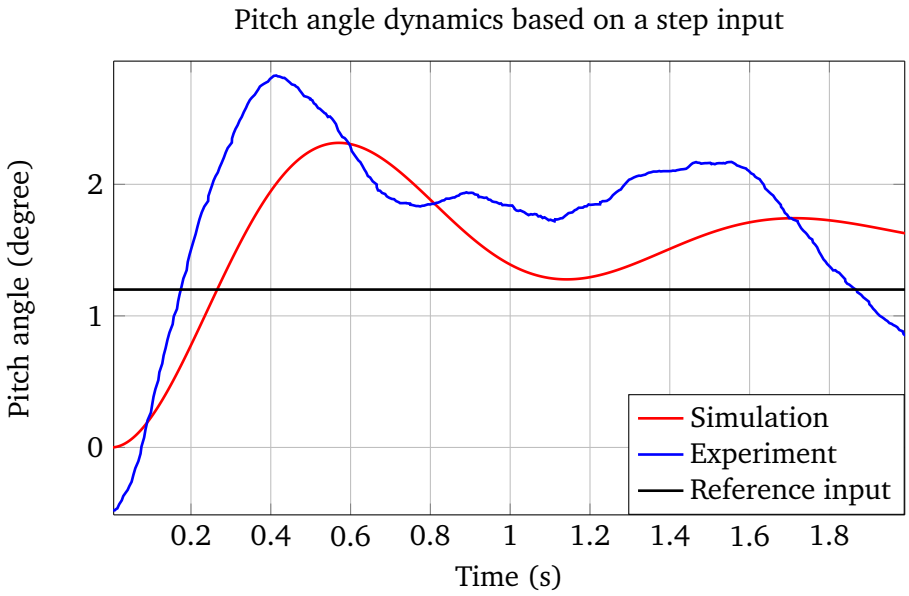


Figure 4.5: Comparison of simulation and experimental data of pitch angle dynamics

Differences between pitch angle and roll angle dynamics were neglected.
In the case of the step input the system shows a very fast rising time (time to get from 10% to 90% of the steady state value) and a reasonable overshoot. Moreover, the Ar.Drone identification highlights the huge DC gain of the system. Throughout the identification process of the Ar.Drone the DC gain varies between $2-3$ with a mean value of 2.2. The gain depends on the actuating variable and decreases with an increasing input. Table 4.2 shows the mean values for the identification of the pitch and roll dynamics

Property	Mean value
Overshoot	20%
Rise time	$0.3s$
Steady-state value	2.64
DC gain	2.2

Table 4.2: characteristics of average pitch and roll angle for an input of $u = 0.1$ $(1.2°)$

for a reference input of $u = 0.1$. This leads to the following transfer function:

$$G_{tilt}(s) = \frac{124.64}{s^2 + 5.06s + 55.15} \tag{4.19}$$

The validation with flight data points out that the fit of the transfer function with the actual dynamics is as low as 50%.
Therefore, we carried out further investigations on the identification of the key transfer function. We flew manually, instead of using a predefined step input. Flying manually aims to stimulate higher frequencies in the system to ensure a better identification of the key parameter. Again the LSE was used to obtain the key parameters. The parameter vector and measurement vector are:

$$\begin{aligned} \psi^T &= [-y(t)^1, -y(t)^2 \quad | \quad u(t)] \\ \hat{\theta}(t) &= [\frac{d_2}{d_0}, \frac{d_1}{d_0} \quad | \quad \frac{n_0}{d_0}] \end{aligned} \tag{4.20}$$

This approach leads to slightly better results with a fit of up to 58%. The new transfer function is as follows:

$$G_{tilt}(s) = \frac{17.99}{s^2 + 1.158s + 12.22} \tag{4.21}$$

The DC gain is comparably small by virtue of strong decrease in the DC gain for faster dynamics that are present in the case of higher values for the actuating variable. The rising time and overshoot are in both cases identical. There are several reasons for the poor performance of the identification procedure but they can be summarized by following points:

1. the coupling effect of the pitch and roll angle has a pivotal role in the identification but it is neglected due to the small angle assumptions in the model. Furthermore, the x-type architecture of the ardrone boosted the coupling effect compared to a cross-type configuration [20].
2. we have no knowledge about the gains of the PD- and P-controller of the attitudes closed loop and we have no information about the design of the higher level controller which guarantees stability.
3. the DC gain fluctuated strongly through our experimental flights. Hence, only a rough estimation is possible.
4. The system acts very sensible to external disturbances such as wind gusts. Moreover, the progression of the pitch and roll angle highly depend on the initial state. Even minor changes within the hover regime lead to different behavior as shown in figure 4.6.
5. A higher order modeling of the system has not delivered better results.

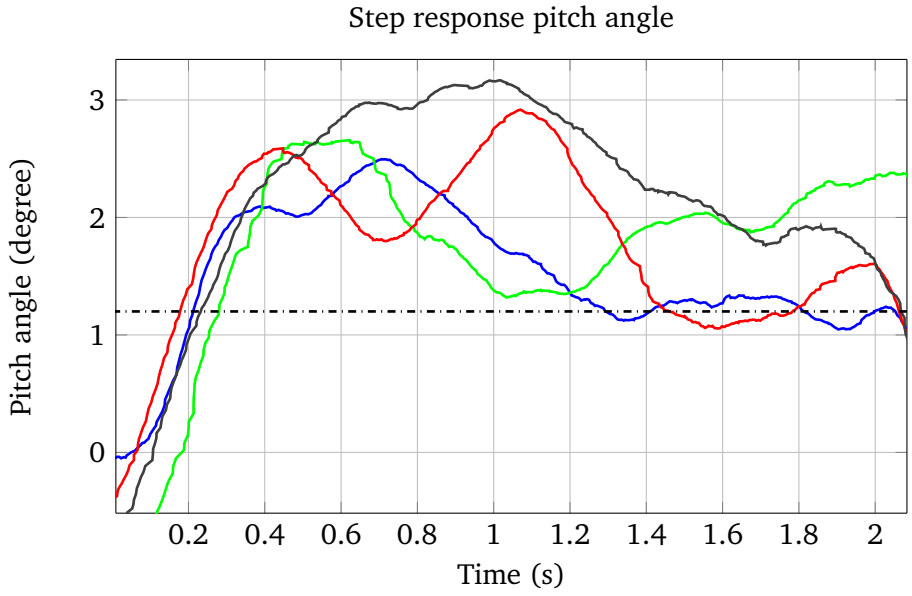


Figure 4.6: The step responses of the pitch angle dynamics are plotted. All progressions start in the hover regime. The reference value is $u = 0.1$ or 1.2° and is plotted as black dotted line. The continuous lines show the step responses.

Identification of translational dynamics

The identification of the lateral translational dynamics is very important for a accurate estimate of the velocities. Therefore, we investigate two different approaches and compare them in this section. Both

approaches are based on a physical model and depned on the tilt angle identification with the reference input θ_{ref}, ϕ_{ref}. The transfer functions describes the relationship between the reference input and the output v_x and v_y, instead of using θ and ϕ as input. Hence, the final transfer function is a third order model.

Usual interpretation

In case of the usual interpretation, derived in section 4.2.3, no identification is carried out, since it is adapted from the attitude identification with an additional integrator $G(s) = \frac{g}{s}$. This leads to the following transfer function:

$$G_{lat}(s) = \frac{v_x}{\theta_{ref}} = \frac{gn_0}{s(s^2 + sd_1 + d_0)} = \frac{424.43}{s^3 + 2.81s^2 + 32.27s} \tag{4.22}$$

Experiments indicate the following restrictions of the usual interpretation:

1. if flying at a constant velocity v_x is considered, the tilt angle θ is not zero but approximately proportional to v_x.
2. if flying at a constant tilt angle θ is considered, the velocity v_x does not grow unbounded, but reaches a steady value approximately proportional to θ.

The experimental results demonstrate that the usual interpretation does not model the dynamics very well. The calculated velocity has a huge drift over time due to the fact that the dynamics are only modeled by an integral term (compare figure 4.7). The assumption to neglect the air drag is reasonable due to the fact that the velocity behaves linear at slow speeds. Hence, the error between experiment and simulation is caused due to another reason.

Revisited interpretation

The revisited model has an unknown parameter f_1 that is identified with the LSE:

$$\begin{aligned} \dot{v}_x &= g\theta - f_1 v_x \qquad f_1 = \frac{k_1}{m} \\ \boldsymbol{\theta} &= [f_1] \qquad \boldsymbol{\psi}^T = [-a_x \quad | \quad \theta] \end{aligned} \tag{4.23}$$

The LSE results in $f_1 = 0.37$ and indicates that the measured acceleration is directly connected to the acceleration of equation 4.23. The measurements indicate that it is only sensitive to translational accelerations in the x- and y-plane[26]. Moreover, the gravitational acceleration, measured in the body-frame, has no influence. The revisited method is drift free over time. This leads to the transfer function:

$$G_{lat}(s) = \frac{v_x}{\theta} = \frac{9.81}{s + 0.37} \tag{4.24}$$

In total the transfer function consists of two parts, namely the lateral translational dynamics $G_{lat}(s)$ and the tilt angle dynamics $G_{tilt}(s)$. The combined model is:

$$G_{vel}(s) = \frac{v_x}{\theta_{ref}} = \frac{n_0 g}{s^3 + s^2(f_1 + d_1) + s(d_1 f_1 + d_0) + d_0 f_1} = \frac{36.96}{s^3 + 1.528^2 + 12.65s + 4.521} \tag{4.25}$$

The results from our experiments indicate that the validation of revisited model has a fit of 80%. Although, it consists of the tilt angle identification. This illustrates that the part $-f_1 v_x$ has the most influence on the acceleration. Furthermore, the results indicate that blade flapping induces a force proportional to the velocity. This force is not negligible. Both approaches show an offset in the velocity till $t = 40s$, because in the hover regime the measured tilt angle is very small and thus leads to imprecise measurements. In the revisited model the offset dissipates as soon as the step input is present (around t $= 40s$). The usual approach continues to drift further away (Figure 4.7).

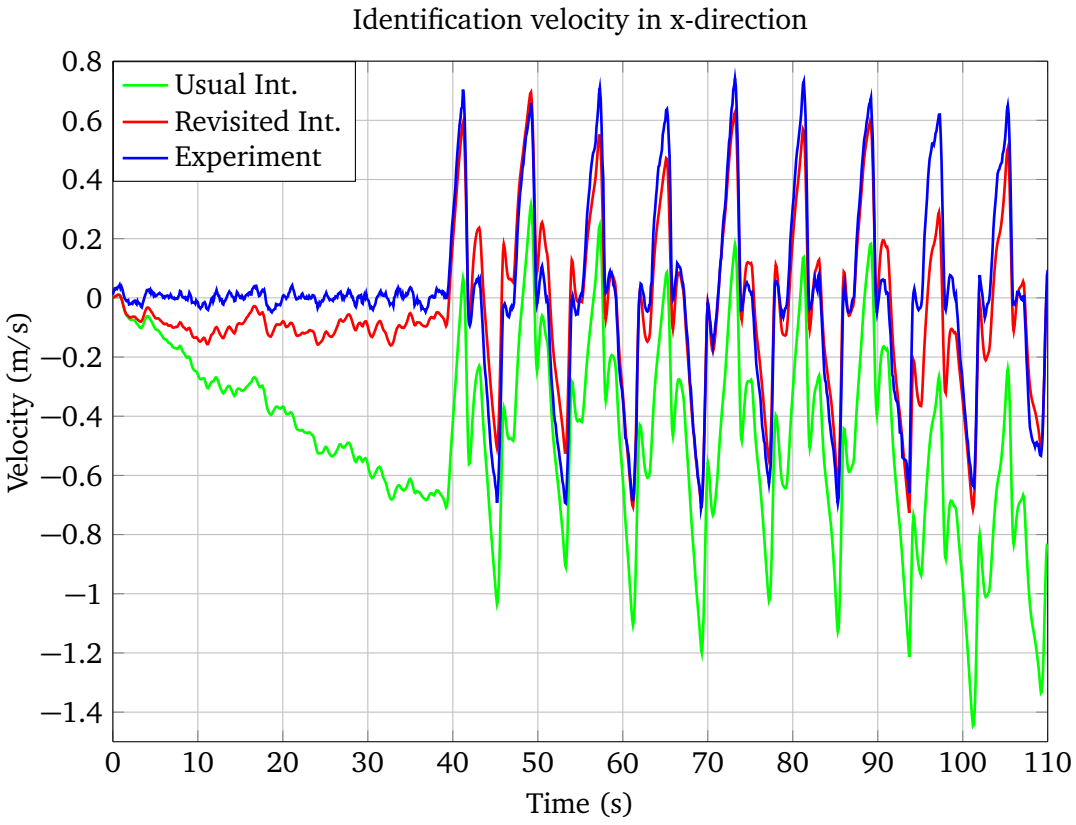


Figure 4.7: Comparison of both approaches with real flight data. The step responses for θ_{ref} $\epsilon[-0.1, 0.1]$ are plotted. The step input starts at time: 40s. The experimental data is marked in blue. The revisited model is marked in red and the usual interpretation is marked in green. The usual interpretation has a drift of the velocity over time.

Identification yaw and height dynamics

The identification between yaw and height dynamics are similar. Both dynamics behave linear and are modeled as first order system.

Height dynamics identification

The dynamic is linear and has no noticeable overshoot. The overshoot is shown in figure 4.9 and is caused due to external disturbances. The key parameter of the height dynamics are shown in table 4.8:

The height dynamic is modeled as simple integrator:

$$G_{height}(s) = \frac{620}{s} \tag{4.26}$$

The validation proves that the height dynamic is linear and is reasonably modeled by a first order system. The fitting of the model with experimental data is around 95%.

Yaw dynamics identification

The key parameter for the yaw dynamic are shown in table 4.10. The dynamic is again modeled as simple integrator with a gain:

$$G_{yaw}(s) = \frac{78.58}{s} \tag{4.27}$$

The validation shows that the model as fit of over 90%.

Property	Mean value
Overshoot	3.5%
Rise time	0.8s
Steady-state value	0.73
DC gain	1.05

Figure 4.8: Key parameter of height dynamics identification. For the identification a step input was used with a length $t = 1s$ and a maximum actuating value of $\dot{z} = 0.7$.

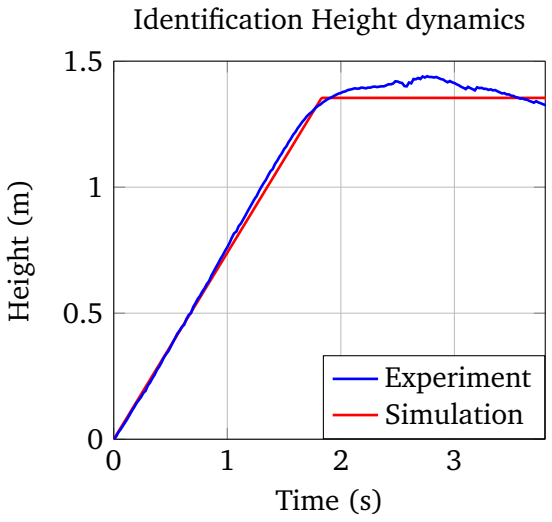


Figure 4.9: Step input of the length $t = 1.8s$ with the maximum actuating value of $\dot{z} = 0.7m/s$.

Property	Mean value
Overshoot	0%
Rise time	0.9s
Steady-state value	100°
DC gain	1.0

Figure 4.10: Key parameter of yaw dynamic identification. The identification was carried out with a step input of the length of 1.2s and the maximum actuating value of $\dot{\psi} = 100°/s$

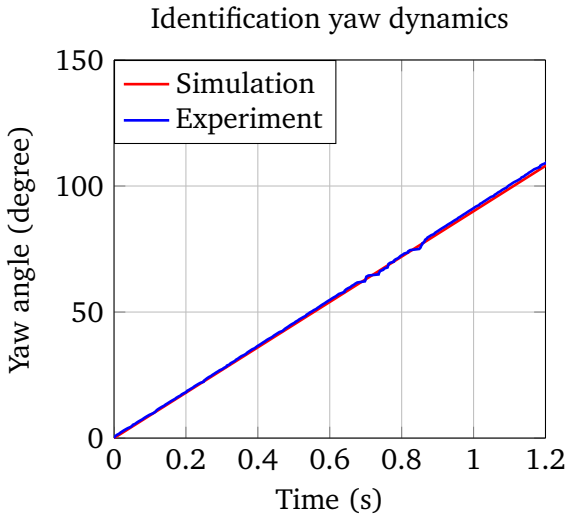


Figure 4.11: Step input of the length $t = 1.2s$ with the maximum actuating value $\dot{\psi} = 100°/s$.

The yaw angle is wrapped in values of $\epsilon[-180°, 180°]$. Hence, it switches the sign during one rotation in clock direction. The rotation starts at 0° then it goes 180°. At that point it instantly switches to $-180°$ and then goes back to 0°.
Furthermore, both identifications share the same problem that the system outputs drift over time. Thus, the identification focus on the transient behavior between two steady states.

4.3.1 Evaluation of the Identification

In this work several dynamics of the quadcopter were identified by mainly using step inputs. The dynamics of the attitude and translational velocities are coupled, and the yaw angle dynamics as well as the height dynamics are decoupled from the rest of the dynamics. We evaluate the result of each identification:

1. **Attitude dynamics:** the attitude dynamics are the center of the identification because they receive the actuating variables and induce translational velocities. The onboard controller of the quad-

copter heavily influences the mechanical dynamics of the tilt angles. Furthermore, we assumed that the dynamics between the x-axis and y-axis are decoupled. This is a flaw in the modeling of the quadcopter and leads to poor identification of around 50% results. The dynamics between the axes are highly coupled and each axis has a nonlinear dynamic. Apart from that the DC gain fluctuates strongly between 2-3. Other researchers performed an identification of the whole quadcopter and they achieved only a fit of around 50%, too [20].

2. **Translational dynamics:** the translational velocities are controlled by adjusting the tilt angle of the quadcopter. Thus, the dynamics between velocity and attitude are coupled. To identify a good model of the dynamics two approaches were investigated. The revisited model includes the dynamics of blade flapping and are assumed to be a repulsive force proportional to the velocity of the quadcopter. The translational dynamics are modeled very precisely even though the attitude dynamics are modeled poorly. The evaluation of the revisited model reached a fit of up to 80% whereas in the traditional approach by modeling the equilibrium state around hover regime a huge drift over time is present.

3. **Height dynamics:** the height dynamic behaves linear and was identified by performing a series of step inputs. The dynamic is modeled with a fit of around 90%. However, measurements drift over time. In addition, the height is hold constant by the internal controller of the Ar.Drone and thus the actuating commands are not available at the ground station. In conclusion, the model is very precise but a ground truth velocity measurement is required to use height dynamics.

4. **Yaw dynamics:** the yaw dynamic is separated from the other attitude dynamics due to the fact that the yaw dynamics does not influence the other dynamics. Moreover, the yaw angle measurement has a huge drift over time of around $4° \frac{1}{min}$. The yaw angle measurement is based on the magnetometer of the quadcopter. The identification leads to every good results of a fitting rate over 90%. Due to the fact that the measurements drift over time, a ground truth yaw angle measurement is required.

5 Control and Data Fusion

The Parrot Ar.Drone 2.0 itself is a fast and stable system without a time delay. However, the closed loop with a high task controller operated at the ground station computer induces a data transport over a wireless LAN. This network introduces time delays and packet losses. Thus, the closed loop system is a time delayed system. Therefore, we first describe the features and behavior of time delayed systems in section 5.1. In section 5.2 we explain different concepts for networked control systems (NCS) that are developed to handle time delays in these systems. In this work, we focus on a predictor based network control scheme to compensate the inherent time delays of the system. Therefore, section 5.2.1 explains the basic principles of the Kalman Filter (KF) and Extended Kalman Filter (EKF). Based on the networked control system a velocity controller is developed to control the lateral translational velocities v_x, v_y. The design of a nonlinear digital PID controller is explained in section 5.3.

5.1 Time Delayed Systems

Time delayed systems (TDS) are of strong interest for the research due to the fact that time delays are present in all real world systems. The first investigations to control time delayed systems started in the late 50s. But the main progress was made in the last decades due to the interest of the digital implementation of the controller. In general a time delayed system is defined as follows [29]:

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{A}_0\mathbf{x}(t) + \mathbf{A}_1\mathbf{x}(t-h_0) + \mathbf{B}_0\mathbf{u}(t) + \mathbf{B}_1\mathbf{u}(t-h_1) \\ \mathbf{y} &= \mathbf{C}_0\mathbf{x}(t) + \mathbf{C}_1\mathbf{x}(t-h_2) \end{aligned} \tag{5.1}$$

Time delayed system are separated into different groups:

1. **State delay:** with $\mathbf{A}_1 \neq 0$. In this case the next state is based on the actual and a memory of old states up to h_0 back in time.
2. **Input delay:** with $\mathbf{B}_0 = 0$ and $\mathbf{B}_1 \neq 0$ the input is based on a delayed actuator. This means that the input signal u(t) is applied by a delay of h_1 at the system.
3. **Output delay:** in the presence of a output delay the actual state $\mathbf{x}(t)$ is not measurable. Thus, only an old measurement delayed h_2 in time is available. In this case $\mathbf{C}_0 = 0$ and $\mathbf{C}_1 \neq 0$.

The influence of a time delay to a system can vary. It may induce a poor performance to the system, or even lead to instability of a stable system. However, it can also induce stability to the system based on a reduction of dynamics. Hence, time delayed control (TDC) is often used by non-delayed systems to add a voluntary small delay h to the control signal to estimate disturbances and to compensate them [30, 31].
If a system has an input delay the pair $\{\mathbf{A}_0,\mathbf{B}_0\}$ is not controllable, therefore the controller is designed in such way that the pair $\{\mathbf{A}_1,\mathbf{B}_1\}$ is controllable. In general it is more difficult to prove the stability of a time delayed system since time delay can add infinite number of roots to the system. When a system is represented as transfer function the different delays of input and output are summed up and the total delay is defined as dead-time e^{-Ts} of the system [29]:

$$G(s) = \frac{Ke^{-Ts}}{\tau s + 1} \tag{5.2}$$

Time delays have different features and are separated into the following categories:

1. **constant delay**
   a) commensurate: the time delay $h_i \epsilon R$ is a minimal multiple of the delay h, if $\frac{h_j}{h_i}$ is rational.
   b) not commensurate: if the time delay are not rational dependent.
2. **time-varying delay**
   a) time bounded delay: $0 < \tau_1 < h < \tau_2$. The time delay h is as a lower bound τ_1 and a upper bound τ_2
   b) arbitrary time delay: $0 < h$. The time delay is unbounded and unknown
3. **sample period:** time delay is split in two groups smaller than a sample period $0 < h < \delta$ and bigger than a sample period $h > \delta$

Moreover, the expression quenching defines a system that is stable if the time delay is constant within known bounds but is unstable if the time delay is varying within these bounds [32, 31]. The different categories of delays can also be interfered with each other. For example a time-varying delay can be bounded with an upper bound less than one sample period.

5.1.1 Control of time delayed systems

The control of time delayed systems is still an open problem in research. Most control concepts are based on approximation of the system. The main goal of control concepts for time delayed systems is to ensure the stability of the system and to compensate the time delay. To compensate the time delay different methods are used. They are split into two groups: The first class of algorithms deals with linear systems with input or output delay and no state delays $\mathbf{A}_1 = \mathbf{0}$. The second class of algorithms deals with state delays $\mathbf{A}_1 \neq \mathbf{0}$.
Now, we explain algorithms of the first class. Those methods do not consider both input and output delays at the same time. However, the output delay can be transformed into an input delay, since the control signal is based on the last available output. A new input can only be calculated if a new output is available. Furthermore, most algorithms use a prediction of the control variable to compensate the time delay.
The most famous method is the smith-predictor [30, 29, 22]. It is a model-based control algorithm that overcomes the impact of time delay. It models the plant as first order model with a dead-time. The predicted output is used for the control. But the prediction is not error free. Thus, the dead-time is used to reduce the error of the prediction in such a way that the predicted output is delayed till it is aligned in time with the output of the plant. The difference of both $\mathbf{e}_k = \mathbf{y}_{k-h} - \mathbf{y}_k$ is added to the predicted output $\mathbf{y}_{k,cor} = \mathbf{y}_k + \mathbf{e}_k$. The usage of this structure reduces the error and thereby improves the control performance. Modifications of the smith-predictor are observer-predictor algorithms (OP-A) such as a Kalman Filter in combination with a discrete predictor.
Another control technique is the finite-spectrum-assignment (FSA) that works in the frequency space. The literature indicates that the digital implementation of FSA can raise numerical problems. Furthermore, the FSA approach requires a constant time delay [30, 16]. The researchers in [33] used a Lambert W function to compensate the time delay of the Ar.Drone. However, they assumed a constant time delay. Other approaches are the H_{inf} and robust control. In a discrete time implementation the H_{inf} handles input delays by using system augmentation [21, 34]. System augmentation transforms a time delayed system into delay-free system. This is only possible if: the time delay is bounded and not too high, given that for every delayed time step of the discrete system a new state is added to the augmented system. However, this leads to an increased number of states and thereby to higher computation costs. This is in special the case for large delays or multiple delays. Therefore, in [21] a new approach is described converting the problem into an optimization problem in the Krein space. In general the H_{inf} is a smoothing estimator and predictor.

The second class of algorithms are the most common approaches H_{inf} techniques and the Popov theory. Both do not consider input or output delays $\mathbf{B}_1 = \mathbf{0}$ [30]. For the sake of clearance all methods are direct or implicit state-predictors to compensate the time delay of the system.

5.2 Networked Control System

The time delay in our closed loop system is mostly introduced by communication time over a wireless LAN the computation time of the ground station computer, and the processing time of the actuator and sensor. The control concepts for time delayed systems (details in section 5.1) are applied on the infrastructure of a wireless LAN. This leads to a networked control system (NCS) [35, 36]. The NCS is displayed in figure 5.1 and consists of 4 parts: the first part is the ground station computer, the second part is the actuator that receives the control command over the NCS, the third part is the plant, and the fourth part is sensor that measures the output and sends it to the GS.
The ground station computer represents a high level task planner with the controller for the closed loop. The ground station sends control commands and receives the responses of the system. The actuator receives the control commands from the GS and performs the control action. The actuator is on-side of the plant and usually consist of a internal controller. The actuator is the input of the plant. The plant is reacting on the changes of the actuator and adapting its state vector $\mathbf{x}_k$. The sensor represents the measurement unit and sends the measurable states $\mathbf{y}_k$ with a predefined sample rate to the GS. As soon as a new measurement is available the ground station calculates a new control command. Then the whole process repeats. In conclusion, the networked control system is a closed-loop with the controller running at the ground station. Hence, it is important to guarantee of a stable connection between GS and the plant. Furthermore, the compensation of time delay is important to ensure the stability of the closed-loop.

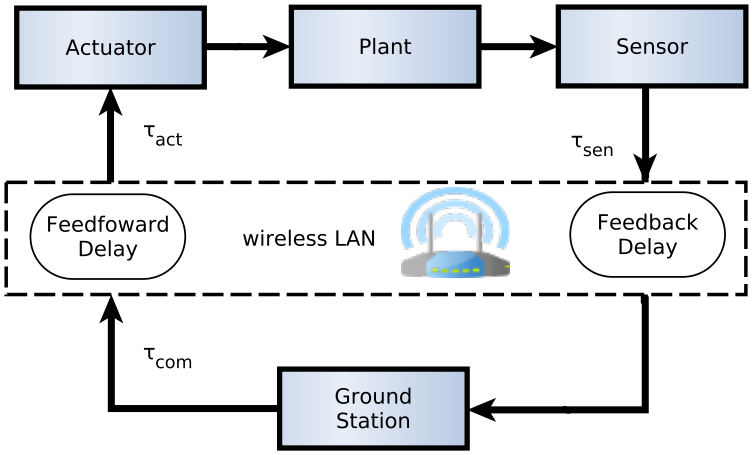


Figure 5.1: This flow chart represents the general structure of a networked control system. A network control system consists on one side of a ground station computer with a controller and an application and on the other side of an actuator, the plant, and a measurement unit.

The networked system in figure 5.1 highlights several time delays. The forward delay τ_{act} is defined as the time to transfer a control signal $\mathbf{u}_k$ from the ground station to the actuator plus the time of the actuator until the desired control command $\mathbf{u}_{k-h_1}$ is active. The total forward delay is h_1 time steps. The feedback delay is defined as time duration until the output states of the plant $\mathbf{y}_k = \mathbf{C}\mathbf{x}_k$ are available at the ground station computer $\mathbf{y}_k = \mathbf{C}\mathbf{x}_{k-h_2}$. The feedback delay consists of two parts. The first component of the time delay is the time duration to perform the measurement plus the delay given through the sample rate of the measurement unit. The second component is the transport delay within the network. Both delays combined are τ_{sen}. The last delay is caused by the ground station itself. The delay τ_{com}

represents the computation time of the ground station. The forward delay and the backward delay do not have the same size, and the effect on the system performance is not the same either. The total delay of the network control system can be expressed as $\tau_{k,tot} = \tau_{k,act} + \tau_{k,sen} + \tau_{k,com}$.
A detailed look at the transport time delays in a wireless LAN conduct that in case of fast system dynamics the transport delay has a significant influence to the overall performance of the system. The wireless network delay is defined through special characteristics: first the time delay is bigger than a sample time $\tau(k) > \delta_k$, it is time-varying, arbitrary and unbounded $\tau(k) > 0$. Furthermore, it is not deterministic, in fact the overall transport delay has nonlinear dynamics [35, 37, 38]. Another challenge is the sudden loss of packages which are caused by the wireless transport. The latency and availability of a stable connection varies due to traffic, jitter, and the position of transmitter and receiver. In the literature a common approach to tackle package loss is the use of play back buffers. The first buffer is placed before the actuator. This buffer stores the next control inputs of the system. A second buffer is placed at the output. During a bad connection the actuator continues working until the first buffer of $\mathbf{u}$ is empty. The second buffer stores all outputs $\mathbf{y}$ and sends them to the ground station if the connection becomes stable again. A different technique is to transfer the the plant into a stable idle state in case of a bad connection.

5.2.1 Kalman Filter and Extended Kalman Filter

In this section, the Kalman filter (KF) and the non linear Extended Kalman filter (EKF) are summarized. The KF is an optimal filter to estimate the state vector $\mathbf{x}_k$ of a linear system. It was invented in the 1950s and gained increasing interest in tracking, localization and navigation tasks. Moreover, the KF allows the incorporation of a variety of different sensors. Thus, it is a data fusion algorithm. The numerous use cases and the convenient implementation for on-line real time processing turned the Kalman filter into a popular algorithm for state estimation. The Extended Kalman filter is based on a non linear process model and is not an optimal estimator.

Kalman filter (KF)

The Kalman filter (KF) is a recursive estimator for linear systems and belongs to the Bayesian filters. The KF is based on a static process model of the dynamic system and explicit computes the belief for the state vector $\mathbf{x}(t)$ and the process covariance $\mathbf{P}(t)$ at time t. The observations $\mathbf{z}(t)$ are provided based on a model of the sensors. The KF is a technique for filtering and prediction of linear systems. The gain $\mathbf{W}_k$ of the KF are chosen to ensure, that the estimate $\hat{\mathbf{x}}(t)$ minimizes the mean-squared-error [39, 40].

Preliminaries

The linear system is described as follows:

$$\mathbf{x}_{k+1} = \mathbf{F}_k x_k + \mathbf{G}_k \mathbf{u}_k + \boldsymbol{\varepsilon}_k \tag{5.3}$$

The state vector $\mathbf{x}_k$ and the input vector $\mathbf{u}_k$ are a vertical vector of size $\Re^{n\times 1}$. $\mathbf{F}_k$ is the state transition matrix of size $\Re^{n\times n}$ and $\mathbf{G}_k$ is the input matrix of size $\Re^{n\times m}$. The process noise $\boldsymbol{\varepsilon_k}$ is Gaussian noise, white and zero mean. The covariance of $\boldsymbol{\varepsilon_k}$ given through $E[\boldsymbol{\varepsilon_k}\boldsymbol{\varepsilon}'_{\boldsymbol{k}}] = \mathbf{Q}_k$.
The measured states are described with the measurement vector:

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \boldsymbol{\delta_k} \tag{5.4}$$

$\mathbf{z}_k$ of size $\Re^{p\times 1}$ and the output matrix $\mathbf{H}_k$ is of size $\Re^{p\times n}$. The measurement noise $\boldsymbol{\delta_k}$ is a Gaussian white noise with zero mean. The corresponding covariance matrix is $E[\boldsymbol{\delta_k}\boldsymbol{\delta}'_{\boldsymbol{k}}] = \mathbf{R}_k$.
Furthermore, the process and measurement variables follow a Gaussian noise distribution. The KF is an optimal estimator for the belief of the current state $\mathbf{x}_k$ as well as for the belief of the process covariance $\mathbf{P}_k$.

Algorithm

The KF algorithm consists of a prediction and a correction cycle. The time discrete implementation of the KF uses a cycle with two time steps. The continuous time $t = kT$ is replaced with the time step k and the step size T [41].

Prediction: First a belief for the next state is computed based on the control input $\mathbf{u}_k$ and the belief of the current state $\mathbf{x}_{k|k}$.

1. State prediction:

$$\mathbf{x}_{k+1|k} = \mathbf{F}_k \mathbf{x}_{k|k} + \mathbf{G}_k \mathbf{u}_k \tag{5.5}$$

2. Measurement prediction:

$$\mathbf{z}_{k+1|k} = \mathbf{H}_k \mathbf{x}_{k+1|k} \tag{5.6}$$

3. Process covariance prediction:

$$\mathbf{P}_{k+1|k} = \mathbf{F}_k \mathbf{P}_{k|k} \mathbf{F}_k' + \mathbf{Q}_k \tag{5.7}$$

Update: In the next time step $k+1$ the difference between the belief of the observation $\mathbf{z}_{k+1|k}$ and the actual observation $\mathbf{z}_{k+1}$ is termed residual $\mathbf{v}_{\mathbf{k+1}}$.

1. Observation residual:

$$\mathbf{v}_{k+1|k} = \mathbf{z}_{k+1} - \mathbf{z}_{k+1|k} \tag{5.8}$$

2. Update state estimation:

$$\mathbf{x}_{k+1|k+1} = \mathbf{x}_{k+1|k} + \mathbf{W}_{k+1} \mathbf{v}_{k+1} \tag{5.9}$$

Additional calculations: During the process of the algorithm additional matrices are calculated. The Filter gain $\mathbf{W}_{k+1}$ of the KF is calculated based on the process covariance $\mathbf{P}_{k+1|k+1}$ and the observation prediction covariance $\mathbf{S}_{k+1}$.

1. Filter gain:

$$\mathbf{W}_{k+1} = \mathbf{P}_{k+1|k} \mathbf{H}_{k+1}' \mathbf{S}_{k+1}^{-1} \tag{5.10}$$

2. Observation prediction covariance:

$$\mathbf{S}_{k+1} = \mathbf{H}_{k+1} \mathbf{P}_{k+1|k} \mathbf{H}_{k+1}' + \mathbf{R}_{k+1} \tag{5.11}$$

3. Update process covariance:

$$\mathbf{P}_{k+1|k+1} = \mathbf{P}_{k+1|k} - \mathbf{W}_{k+1} \mathbf{S}_{k+1} \mathbf{W}_{k+1}' \tag{5.12}$$

Furthermore, the filter gain $\mathbf{W}_{k+1}$ is calculated by solving the Ricatti equation for a time invariant system. The solution converges to a steady finite value if and only if the set {**F**,**H**} is fully observable. Thus, the filter gain **W** is static and is calculated off-line [41, 42].

Extended Kalman Filter (EKF)

The extended Kalman filter (EKF) is a form of the KF and should be employed if the process and/or the observation is non-linear. The process matrices $\{\mathbf{F},\mathbf{G}\}$ and the observation matrix $\mathbf{H}$ are replaced by arbitrary functions f and h:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k,\mathbf{u}_k,\varepsilon_k) \tag{5.13}$$

$$\mathbf{z}_k = h(\mathbf{x}_k,\delta_k) \tag{5.14}$$

Hence, the belief is no longer Gaussian and therefore the solution of the EKF has no closed form. This causes the result to an approximation of the true belief, not exact as it is when using the KF. Despite that the assumptions are the same as in the KF and stay valid.
For the implementation of the EKF the nonlinear functions f and h are linearized. To do so, the EKF uses a first-order Taylor expansion:

$$f'(\mathbf{x_k},\mathbf{u_k}) := \frac{\partial f(\mathbf{x_k},\mathbf{u_k})}{\partial \mathbf{x}_k} \approx f(\overline{\mathbf{x}}_k,\mathbf{u}_k) + \underbrace{\frac{\partial f((\overline{\mathbf{x}}_k,\mathbf{u}_k)}{\partial \overline{\mathbf{x}}_k}}_{:=\mathbf{F}_k}(\mathbf{x}_k - \overline{\mathbf{x}}_k) \tag{5.15}$$

The nonlinear function is linearized around the point $\overline{\mathbf{x}}_k$ and exploration is achieved by a proportional term to the gradient $f'(x_k,u_k)$. The matrix $\mathbf{F}_k$ is called Jacobian matrix and is of size $\in \Re^{n\times n}$. The same procedure is carried out for the observation function.

$$h'(\mathbf{x}_k) := \frac{\partial h(\mathbf{x}_k}{\partial \mathbf{x}_k} \approx h(\overline{\mathbf{x}}_k) + \underbrace{\frac{\partial h(\overline{\mathbf{x}}_k)}{\partial \overline{\mathbf{x}}_k}}_{:=\mathbf{H}_k}(\mathbf{x}_k - \overline{\mathbf{x}}_k) \tag{5.16}$$

Algorithm

The equations for the KF and EKF are almost identical. We briefly summarize the equations for EKF:

1. **Prediction:**

$$\begin{aligned}
\mathbf{x}_{k+1|k} &= f(\mathbf{x}_{k|k},\mathbf{u}_k) \\
\mathbf{z}_{k+1|k} &= h(\mathbf{x}_{k+1|k}) \\
\mathbf{P}_{k+1|k} &= \mathbf{F}_k\mathbf{P}_{k|k}\mathbf{F}_k' + \mathbf{Q}_k
\end{aligned} \tag{5.17}$$

2. **Update:**

$$\begin{aligned}
\mathbf{v}_{k+1|k} &= \mathbf{z}_{k+1} - \mathbf{z}_{k+1|k} \\
\mathbf{S}_{k+1} &= \mathbf{H}_{k+1}\mathbf{P}_{k+1|k}\mathbf{H}_{k+1}' + \mathbf{R}_{k+1} \\
\mathbf{W}_{k+1} &= \mathbf{P}_{k+1|k}\mathbf{H}_{k+1}'\mathbf{S}_{k+1}^{-1} \\
\mathbf{x}_{k+1|k+1} &= \mathbf{x}_{k+1|k} + \mathbf{W}_{k+1}\nu_{k+1} \\
\mathbf{P}_{k+1|k+1} &= \mathbf{P}_{k+1|k} - \mathbf{W}_{k+1}\mathbf{S}_{k+1}\mathbf{W}_{k+1}'
\end{aligned} \tag{5.18}$$

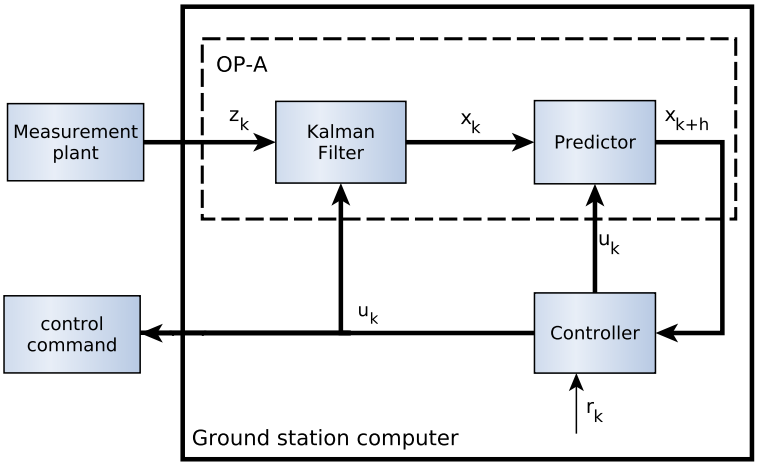


Figure 5.2: The observer-predictor algorithm consists of two parts: a Kalman Filter for data filtering and data fusion, and a h-step ahead prediction scheme to compensate the time delay given the knowledge of the system model. Based on the estimate of the state vector a controller is used.

5.2.2 Predictor based Kalman Filter

The previous section explained the structure of NCS. This section focus on the time delay compensation. As mentioned in section 5.1 most techniques are based on state-prediction. Our approach uses an observer-prediction algorithm (OP-A) [43, 44, 45]. The algorithm consists of two parts, namely the observer and predictor.

The observer is a Kalman Filter (details in section 5.2.1), because the filter receives measurements from different sensors with multiple time delays. The KF has a flexible structure that allows to handle multiple delays. Moreover, the measurements are overlain by noise. Last, the system is prone to packet losses. The KF is able to counteract the issues due to use of a model of the plant. In comparison to a simple low-pass filter the KF adds no additional delay. Merely the computation time adds a minimal delay. A second benefit of the KF setup is the capability to adapt the system to new measurements. In conclusion the KF fuses the data from the different sensors of the plant and filters the noisy measurements. The output of the KF is a filtered estimate of the delayed state vector $\mathbf{x}_k$ of the system. The KF has a prediction and correction cycle. The predictor of the KF is used to predict h-step ahead, instead of one step, and thereby compensates the time delay. This approach is called h-step ahead prediction. To achieve a good prediction result over a long time ahead prediction a good model of the system dynamics is required. The predictor delivers an estimate of the state $\mathbf{x}_{k+h}$ that is used for the feedback controller. The main advantage of the use of a state-predictor is that in the case of an ideal prediction, which is an error free prediction $\mathbf{0} = \mathbf{x}_{k+h} - \mathbf{z}_{k+h}$, the closed-loop behaves as a system without time delay [14, 46, 47].

h-step ahead prediction

Here the h-step ahead prediction scheme is derived from the discrete predictor of the KF. We assume a state space system with an input delay:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_{k-h} \tag{5.19}$$

The next state depends on the actual state and the h time-steps old control input $\mathbf{u}_{k-h}$. Furthermore, the total time delay $\tau_{k,tot}$ is a multiple of the sample time T, so that the input delay is written as $h = \tau_{k,tot}$. This is reasonable due to the fact that a control command is calculated if a new measurement is available.

The last h control commands are stored in the KF and the predictor. In addition the predictor stores the last h predicted states. The equation 5.19 calculates the next sate at time stamp $k+1$ based on the control command $\mathbf{u}_{k-h}$. This equation is successively executed for all available control inputs from $\mathbf{u}_{k-h}$ till $\mathbf{u}_{k-1}$. In total it is executed h times. This leads to the h-step ahead prediction scheme [47, 14]:

$$\begin{aligned}\mathbf{x}_{k+h} &= \mathbf{A}^h\mathbf{x}_k + \mathbf{A}^{h-1}\mathbf{B}\mathbf{u}_{k-h} + \ldots + \mathbf{A}\mathbf{B}\mathbf{u}_{k-2} + \mathbf{B}\mathbf{u}_{k-1} \\ \mathbf{x}_{k+h} &= \mathbf{A}^h\mathbf{x}_k + \sum_{i=0}^{h-1}\mathbf{A}^{h-i-1}\mathbf{B}\mathbf{u}_{k-h+i}\end{aligned} \tag{5.20}$$

Equation 5.20 shows the final prediction scheme. The is used in the output feedback controller.

$$\mathbf{u}_k = \mathbf{K}^T\mathbf{C}(\mathbf{A}^h\mathbf{x}_k + \sum_{i=0}^{h-1}\mathbf{A}^{h-i-1}\mathbf{B}\mathbf{u}_{k-h+i}) \tag{5.21}$$

The predicted state x_{k+h} only depends the last h time steps of the state vector and the input vector. $\mathbf{K}$ is the matrix of the output feedback controller. The prediction based controller has a memory of the last states and control commands. To implement the discrete predictor scheme the total delay has to be known.

5.3 Digital PID control

The velocity controller is a digital implementation of a PID control scheme with nonlinear modifications. The equation of the ideal PID control is as follows [48, 49]:

$$u_k = K_R[e_k + \frac{T}{T_N}\sum_{i=0}^{k-1}e_i + \frac{T_v}{T}(e_k - e_{k-1}) \tag{5.22}$$

The error $e_k = r_k - y_k$ is defined as difference between reference value and the process variable. The parameter: K_R is referred to as gain of the proportional Term. T_N is the gain of the integral term and T_v is the gain of the derivative term. T is the sample rate of the controller. The standard PID controller is modified to add additional degrees of freedom to the controller. Those allow to achieve a better control performance due to the fact that more parameter are available to tune. The modifications are as follows:

Setpoint Weighting

The aim of setpoint weighting is to reduce the overshoot of the PID controller. Furthermore, the impulse of the actuating variable due to an instant change of the set point (step input) is reduced. The simplest implementation is a weighting of the set point of the proportional and the derivative term. Both terms are weighted separately. The new set point $b \cdot r_k$ is a fraction of the original set point r_k. The modification scales the set point. The weighting of the proportional term is b and the weighting factor of the derivative term is c. Both have a value between $b, c \epsilon [0,1]$ [50, 51]. To implement set point weighting the PID controller of equation 5.22 is modified. Instead of using the error e_k, a new weighted error is used. Thus, the error term changes from $(r_k - y_k)$ to $(b \cdot r_k - y_k)$ for the proportional term and to $(c \cdot r_k - y_k - c \cdot r_{k-1} + y_{k-1})$ for the derivative term. This leads to equation:

$$u_k = K_R[(b \cdot r_k - y_k) + \frac{T}{T_N}\sum_{i=0}^{k-1}e_i + \frac{T_v}{T}(c \cdot r_k - y_k - c \cdot r_{k-1} + y_{k-1}) \tag{5.23}$$

Anti-Windup - Conditional Integration

Anti-Windup techniques are used to limit the influence of the integral term at the actuating variable. If the control deviation is huge the integral part starts to sum up the error and while doing so it increases the actuating variable until it is in saturation. If the system is in saturation the closed loop system behaves like an open loop system with a constant input $u = u_{max}$. However, the actuating variable still increases. This behavior is not desired. The following options are implemented to avoid saturation [50]:

1. Limit the integral to a predefined value such that it is a fraction of the maximum actuating variable.
2. The integration should be stopped if the process variable is far away from the set point.
3. The integration should be stopped when the actuating variable is in saturation and the value is hold constant.
4. The integration should be stopped when the actuating variable saturates and the error signal and the actuating variable have the same sign.

Those four rules lead to the following algorithm:

$$u_{k+1} = \begin{cases} u_k + Ke_{k+1} & \text{if} \quad u_k = v_k \quad \text{or} \quad sgn(e_k) \neq sgn(u_k) \\ u_k & \text{if} \quad u_k \neq v_k \quad \text{and} \quad sgn(e_k) = sgn(u_k) \end{cases} \tag{5.24}$$

The Derivative Term

The derivative action provides a phase lead, which counteracts the phase lag created by an integrator. It is also helpful to counteract disturbances. In the digital implementation the derivative term is defined as: $\frac{e_k - e_{k-1}}{T}$ with sample time T. However, the implementation of the derivative term leads to two problems. First, an instant change in the set point leads to an infinite jump of the derivative term. Second, the measurements are overlain by noise, the impact of noise is increased dramatically through the derivation of the short sample time T. To eliminate the first problem the derivative of the process variable is used, instead of the error signal. This feedback is multiplied by a negative unity block. This structure smooth sudden changes of the system created through set point changes or disturbances. The new derivative term is:

$$\text{D:} \quad -\frac{T_v}{T}(y_k - y_{k-1}) \tag{5.25}$$

6 Implementation

In this chapter, we describe the whole setup consisting of the Ar.Drone and the ground station computer. Moreover, we explain the methods we developed and implemented to compensate the time delay, to synchronize the measurements from the different sensors, and to control the translational velocities. The first section shows the complete system as well as the software architecture of the developed solution. Furthermore, it defines the interfaces that are used with the Ar.Drone and other external packages. This allows other researchers to use our developed framework. The second section explains the role of ROS in the course of this thesis. In particular, the used packages are explained and how they fit to the problem statement. Section three is the main section and explains how the observer-prediction algorithm is implemented. Furthermore, this section explains how the data from the different sensors is synchronized, because each sensor has different delays and update rates. Finally, the section points out how the available packages were modified to ensure an optimal result for the task of time delay compensation and data synchronization of the Parrot Ar.Drone 2.0. The last section briefly lists the parameter of the PID controller and describes the structure of the closed loop.

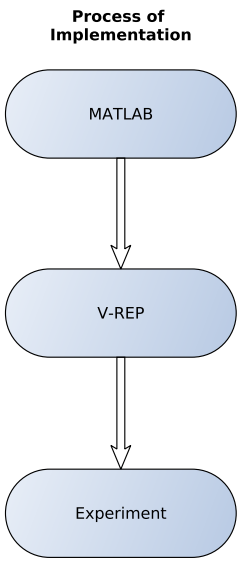


Figure 6.1: Flow chart: implementation

During the process of this thesis, firstly, a concept was developed in MATLAB Simulink using a simplified model identified in chapter 2. The developed algorithm was validated with synthetic data, enriched with Gaussian noise. Moreover, it was validated with experimental flight data. Secondly, the developed algorithms were converted from MATLAB into the C++ and the ROS environment. ROS has a modular structure that allows other researchers to use the framework developed in this thesis. Furthermore, it allows the researchers to use public available packages for a variety of tasks such as communication between the ground station computer and the Ar.Drone. In this work, we partly used public packages, modified them, and committed our modifications to the community. To ensure a successful transfer into ROS, the algorithm was tested in V-REP (virtual robot experimentation platform). V-REP has a nonlinear quadcopter model based on the Parrot Ar.Drone with nonlinear rotor dynamics. However, the internal controller of both models are different. Instead of controlling the tilt angles, the velocity of the air particle which go through the rotor is controlled. Furthermore, the time delay of the closed loop is modeled as part of the communication between V-REP and ROS. By virtue of the non-consisting model of V-REP in comparison to the real quadcopter V-REP was only used as platform to verify how the different packages of our framework interact with each other. Finally, all parameters were determined experimentally and were validated by performing flight maneuvers.

6.1 System Overview and Approach Outline

In this section the components of the developed approach are described. Figure 6.2 displays the system overview.

Quadcopter

The Ar.Drone is a closed system and no modifications were performed. As described in chapter 3 the Ar.Drone has several sensors with different executing rates. All measurements are passed to the Data Processor and are processed internally. No information is available which algorithms are used. However, the following algorithms can be considered: Kalman Filter, Extended Kalman Filter, and complementary filter both linear and nonlinear [2]. For example the velocities estimated from the downward facing camera are fused with the accelerations of the IMU to achieve more precise velocity estimate [2]. Furthermore, they are prone to error due to the low-cost hardware. The processed measurement data are sent to the ground station computer. Hence, only the filtered sensor data is available at the ground station. The filtered sensor data and the control commands from the ground station are forwarded to the inner control loop of the Ar.Drone. The nested architecture is described in 3.1.4. The inner loop controls the tilt, the yaw angle, and the height of the drone. The loop runs fully autonomously on the Ar.Drone. Thus, no additional delays arise. The inner control loop and the data acquisition is executed with $200Hz$. But control commands, are only received to a rate up to $100Hz$ [2, 1].

Software architecture

In this paragraph, the software architecture of the ground station computer is summarized. All processes at the ground station are executed with $100Hz$, because this is the maximal frequency at which control commands are transmitted to the Ar.Drone. The system consists of four components:

1. **Ardrone autonomy:** The node handles the communication between the Ar.Drone and the ground station. The process receives all sensor measurements from the Ar.Drone and converts the received measurements into messages in the ROS environment. The measurements are received without a time stamp, so the time of measurement is unknown. However, the measurements get a time stamp as soon as they arrive at the ground station. Afterwards, they are published as odometry message [27]. An odometry message consists of pose data (position and orientation) and twist data (linear and angular velocity in all three axis). The process also receives the desired control command from the PID controller and transforms the control command (ROS message) into a command executable for the Ar.Drone. Afterwards the commands are transmitted to the drone.

2. **Observer-Predictor Algorithm (OP-A):** the OP-A algorithm is implemented as described in chapter 5.2 and consists of three processes. The first process is a Kalman Filter developed by Charles River Analytics, inc. It receives an odometry message and computes the pose and velocity of the drone. The estimated state $\mathbf{x}_k$ is time delayed and thereby passed to the h-step ahead predictor. To compute the future state $\mathbf{x}_{k+h}$, the predictor needs the estimated current state $\mathbf{x}_k$ of the KF, the last h control commands $\mathbf{u}_k$ and the total time delay of the closed loop $\tau_{k,tot}$. The network delay is continuously measured in a ping process [14]. Based on the data of all three processes the predictor computes the future state $\mathbf{x}_{k+h}$ at which the next control command takes effect.

3. **PID control:** controls the lateral translational velocities v_x, v_y. The yaw angle ψ is not controlled, since no drift free yaw angle measurement is available. The controller calculates the control command based on the predicted velocity of the future state vector $\mathbf{x}_{k+h}$ and the reference velocities $\mathbf{v}_{ref}$. The PID controller sends control command with time stamp to the OP-A process group and to the Ardrone Autonomy process. A PID control scheme of the structure in chapter 5.3 is implemented.

4. **Application:** this process calculates the reference velocities $\mathbf{v}_{ref}$ of the Ar.Drone. In this chapter we do not focus on applications. The aim of this project is to create a framework to compensate the inherent time delays, to synchronize the data, and to control the velocities of the quadcopter. In chapter 7 we introduce an application for our framework, namely, a path following algorithm based on a velocity field in addition to an obstacle avoidance algorithm.

The framework can be used in an already existing environment. The following interfaces are provided:

1. **Input:** the framework only requires the odometry message of the $ardrone\_autonomy$ process.
2. **Output:** the framework publishes a topic called PID/cmd_vel that contains the control command.

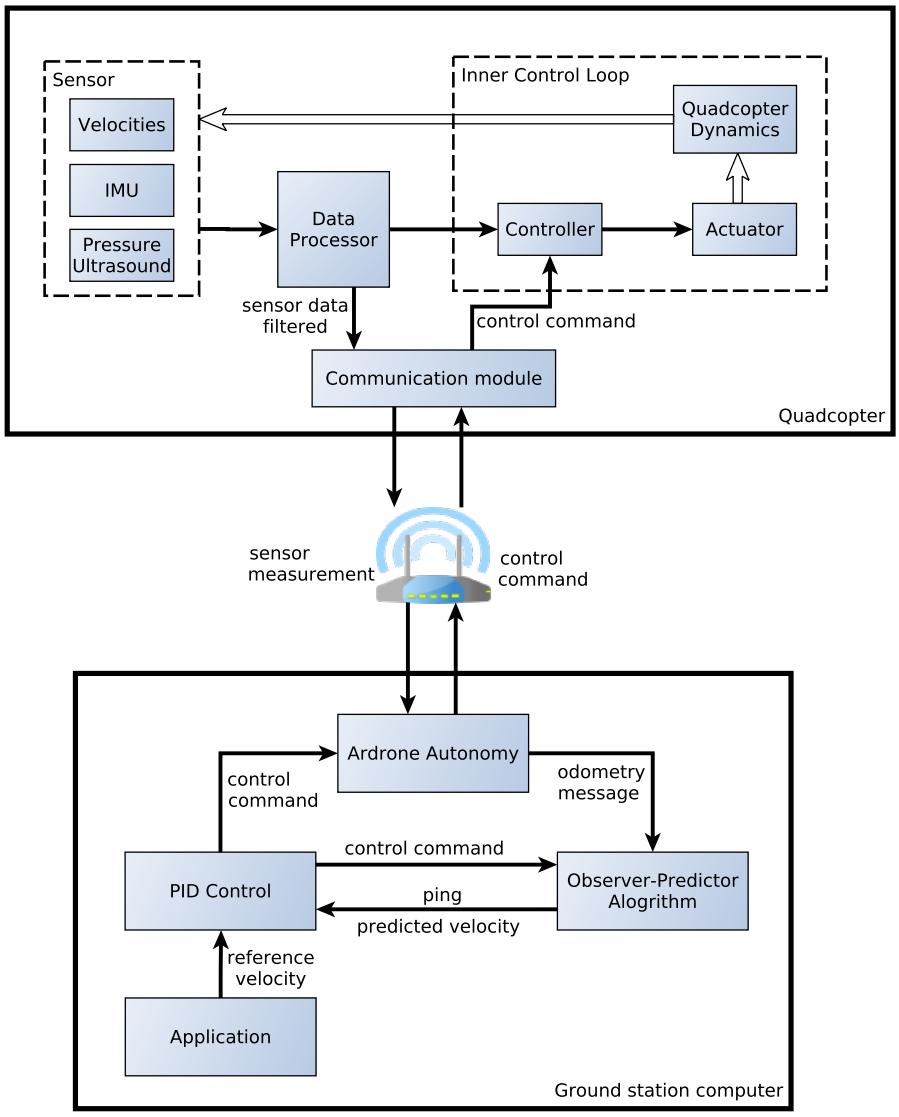


Figure 6.2: The flow chart represents the structure of the whole framework. The system is split intro a ground station computer and a quadcopter. The data is transmitted wireless. The Ar.Drone has a fast not time delayed inner control loop to control the speed of the rotors. The ground station is an outer control loop that receives the sensor data with an inherent time delay. The time delay is compensated and a control command is calculated based on the reference velocity and the time delay compensated sensor data.

6.2 ROS packages

This section abstracts the features of the two public packages that are used in the course of this work. Both packages were modified and the modifications were committed to the developers which added the developed features to their packages [18].

6.2.1 Ardrone Autonomy

The ardrone_autonomy package is based on the official SDK from Parrot. it is the standard package in the ROS environment for sending and receiving data from the Ar.Drone. The package is build and maintained from the autonomy lab at the Simon Fraser University in Canada [27].
The standard package allows to set default boundaries for key features of the Ar.Drone such as tilt angles, maximal velocities or height.

Reading data from the drone

The Ar.Drone is able to send data in real time mode as fast as $200Hz$. Therefore, the *looprate* of the main function of the ardrone_autonomy package should be set to at least at the same rate. Otherwise data will arrive delayed or even get lost. All data is represented according to the ROS REP-103 defined units and the coordinate frame [19]. In general the state of the drone can be separated into the flying state and the landing state. The height and horizontal speeds are only published during the flying state. Among others these are the messages that are published by the drone:

- Navdata (i.e. Battery, tilt angles, velocities, acceleration)
- Odometry data
- Magnetometer data
- IMU data
- Camera video stream
- Tag detection

The navdata is an especially for the Ar.Drone defined package that consists of the information available in the odometry message and additional information related to the state of the drone. In contrast to the odometry package navdata does not follow the REP-103 coordinate frame and SI unit instruction.
The ardrone_autonomy package has originally has not supported to adjust the covariance values of the received measurements. During this work the feature was successfully added the package.

Sending commands to the drone

Control commands can be send to the drone in a rate up to $100Hz$. For basic maneuvers these commands are as follows:

- take off
- land
- reset

are predefined commands. To maneuver the drone the following control commands are published as $geometry\_message :: Twist$:

- $linear.x$ (roll angle)
- $linear.y$ (pitch angle)
- $linear.z$ (height rate)
- $angular.x/angular.y$ (deactivate auto-hover)

- $angular.z$ (yaw angle rate)

If all control inputs are zero the Ar.Drone activates its auto-hover mode. To prevent the drone to go to auto-hover mode the $angular.x$ or $angular.y$ value has to be set to an arbitrary value unequal to zero. All control commands $\mathbf{u}$ are given as fraction of the maximum value and are restricted to ϵ [−1,1].

$$\varphi_{ref} = u * \varphi_{max\_tilt\_angle} \tag{6.1}$$

Equation 6.1 shows the control input for the desired pitch angle. As marked in equation 6.3 the desired tilt angle can be interpreted as a force input.

$$a_x = \sin(\varphi_{ref}) * g \tag{6.2}$$

$$a_y = \sin(\phi_{ref}) * g \tag{6.3}$$

6.2.2 Robot localization

The framework of the observer-predictor algorithm is based on the EKF in robot_localication developed by Charles River Analytics, Inc. This package is widely used in academia and continuously under further development [3]. This package is used instead of others, because:

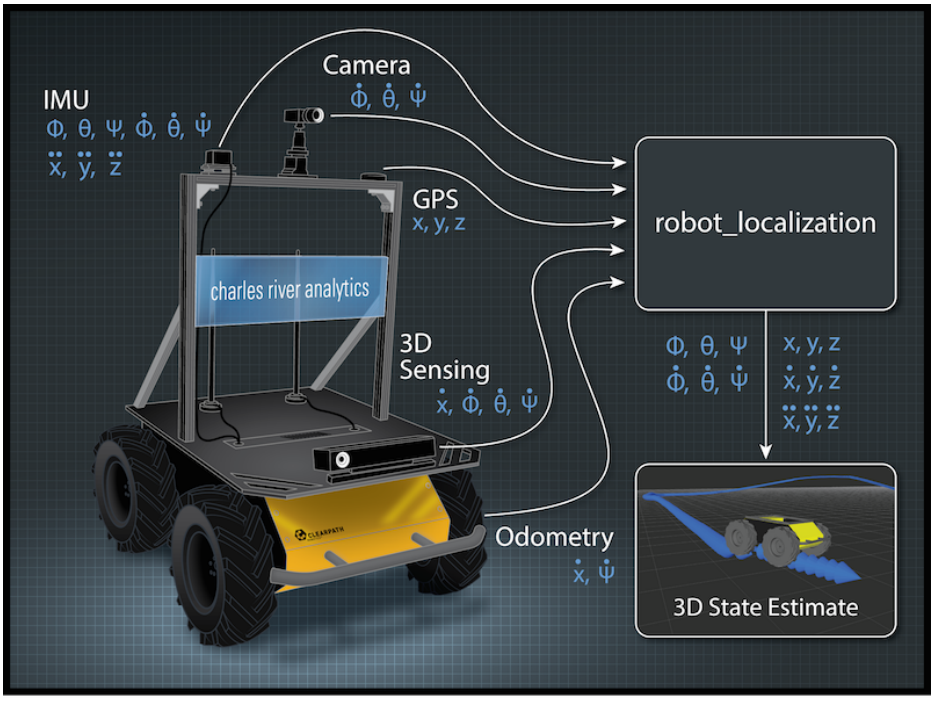


Figure 6.3: The figure presents the different types of sensors that can be fused and the states than can be estimated [3].

1. The package handles an arbitrary amount of sensors
2. It supports a variety of different ROS message types
3. The algorithm is able to estimate the state vector of a robot operating in 3D with all 6DOF
4. The algorithm is able to execute a prediction/correction cycle with the measurements available at the current time

5. The filter has a variable executing rate

Figure 6.3 shows the variety of sensors that can be used in the package. The package is able to estimate in total 15 states:

$$\mathbf{x}^T = [x,y,z,\phi,\theta,\psi,\dot{x},\dot{y},\dot{z},\dot{\phi},\dot{\theta},\dot{\psi},\ddot{x},\ddot{y},\ddot{z}] \tag{6.4}$$

The rotational values are expressed as euler angles. The prediction is based on a model of motion. Here the model of motion is an omnidirectional model of a ground robot derived by the standard Newtonian dynamics. For instance the translational dynamics in the x-direction are:

$$\begin{aligned} &\text{position:} \quad x_{new} = x_{old} + \dot{x} \cdot t + \frac{1}{2} \cdot \ddot{x} \cdot t^2 \\ &\text{velocity:} \quad \dot{x}_{new} = \dot{x}_{old} + \ddot{x} \cdot t \end{aligned} \tag{6.5}$$

The implemented model of motion does not allow us to use an actuating variable. Hence, the term $\mathbf{B}(\mathbf{x}_k,\mathbf{u}_k)$ misses. Next, the implementation is briefly highlighted. It is assumed that $f(\mathbf{x}_{k-1})$ and $h(\mathbf{x}_k)$ are in general nonlinear functions, $\mathbf{w}_{k-1}$ is the process noise, and $\mathbf{v}_k$ is the measurement noise, both noises are normally distributed.

$$\begin{aligned} \mathbf{x}_k &= f(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1} \\ \mathbf{z}_k &= h(\mathbf{x}_k) + \mathbf{v}_k \end{aligned} \tag{6.6}$$

By virtue of the variety of different sensors each sensor is directly correlated to a state variable. Thus, the $\mathbf{H}(\mathbf{x_k})$ matrix becomes the identity matrix. Furthermore, only measured states are updated in the prediction/correction cycle. In detail, the $\mathbf{H}(\mathbf{x_k})$ matrix has only nonzero elements on the diagonal of the states which are measured at the actual cycle. The multi-rate data fusion allows partial updates of the state vector [52].

How to use package

As first step the user has to define to which topics the process should subscribe to receive all necessary measurements. For each topic an update vector is defined to configure which state variables are measured and used in the correction cycle. This allows the user to disable faulty sensor data. Moreover, values of the measurement covariance of each measured state variable are required. The user has to set the:

1. Measurement covariance matrix
2. Process covariance matrix
3. Initial estimate covariance matrix

A covariance value of zero means that the measured or modeled value of the state vector is perfect [52].

6.3 Predictor based Kalman Filter

The goal of this section is to describe the implementation of the predictor based Kalman Filter. Our aim is to estimate a smooth continuously state vector without time delays. The approach is separated into two steps. Firstly, a Kalman Filter estimates the state variables of the developed framework based on the robot_localization package. Secondly, a discrete predictor is developed to counteract the inherent time delay and estimates a future state vector at the time the next control command takes effect. The future state vector is used in the PID control package and is also available for the application process.

6.3.1 KF State Estimation

Here, the state vector of the Kalman Filter and the covariance matrices are defined. The section is split into a prediction and correction paragraph. First, the prediction term is explained, since it allows a better understanding of the filter.

Prediction Model

The prediction model is a model of motion. The model of motion from the robot localization package is based on a ground robot and hence not very precise. For this reason, the model of motion was replaced with the identified quadcopter model of chapter 3. The new model allows the use of a control term. In addition, the EKF becomes a KF. The prediction model consists of 10 states. All state variables are represented in the body-frame also all rotational values are given as Euler angles. The prediction model is:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{6.7}$$

$$\dot{\mathbf{x}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & g & 0 & \frac{-k_1}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & g & 0 & 0 & 0 & \frac{-k_1}{m} & 0 & 0 & 0 & 0 \\ -d_{z0} & 0 & 0 & 0 & 0 & 0 & -d_{z1} & 0 & 0 & 0 \\ 0 & -d_0 & 0 & 0 & 0 & 0 & 0 & -d_1 & 0 & 0 \\ 0 & 0 & -d_0 & 0 & 0 & 0 & 0 & 0 & -d_1 & 0 \\ 0 & 0 & 0 & -d_{\psi 0} & 0 & 0 & 0 & 0 & 0 & -d_{\psi 1} \end{pmatrix} \begin{pmatrix} z \\ \phi \\ \theta \\ \psi \\ v_x \\ v_y \\ v_z \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{z0} \\ n_0 & 0 & 0 & 0 \\ 0 & n_0 & 0 & 0 \\ 0 & 0 & n_{\psi 0} & 0 \end{pmatrix} \begin{pmatrix} \phi_{ref} \\ \theta_{ref} \\ \dot{\psi}_{ref} \\ \dot{z}_{ref} \end{pmatrix} \tag{6.8}$$

Finally, the continuous time system is discretized at a rate of δ_t sample/s. The discrete-time representation is defined:

$$\mathbf{A}_d = \mathrm{e}^{\mathbf{A}\,\delta_t} \qquad \mathbf{B}_d = \int_0^{\delta_t} e^{\mathbf{A}\,\tau} \mathrm{d}\tau\ \mathbf{B} \qquad \mathbf{C}_d = \mathbf{C}. \tag{6.9}$$

The discrete-time representation is implemented through a first order approximation:

$$\mathbf{A}_d = \mathbf{I} + \mathbf{A}\,\delta_t \qquad \mathbf{B}_d = \mathbf{B}\,\delta_t + \mathbf{A}\,\mathbf{B}\frac{\delta_t}{2} \tag{6.10}$$

To obtain good estimation results the process noise covariance matrix $\mathbf{Q}_k$ and the initial estimate covariance $\mathbf{P}_0$ have to be adjusted. Both were tuned experimentally by hand:

$$\mathbf{Q}_k = \begin{pmatrix} 1e^{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e^{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1e^{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1e^{-2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5e^{-5} & 0 & 5e^{-5} & 0 & 0 & 0 & 0 & 0 \\ 0 & 5e^{-5} & 0 & 0 & 0 & 5e^{-5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5e^{-5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1e^{-2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1e^{-2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1e^{-2} \end{pmatrix} \tag{6.11}$$

The values of $\mathbf{Q}_k$ point out that the dynamic model of the lateral translational velocities is more precise than the tilt angle dynamics. This confirms our results from the quadcopter identification. The off diagonal elements of $\mathbf{Q}_k$ show the coupling between tilt angle and lateral velocity. Furthermore, we assume that the process noise covariance matrix is time invariant.
The initial estimate covariance matrix $\mathbf{P}_0$ is simple to tune. The values should not be to small otherwise the solution will not convergent to the true value. Moreover, it is important to model all coupling effects by setting the according element to a value unequal to zero. For the matter of simplicity all elements are set to the same value.

$$\mathbf{P}_o = \begin{pmatrix} 0.256 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.256 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.256 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.256 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.256 & 0.256 & 0 & 0.256 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.256 & 0.256 & 0 & 0 & 0.256 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.256 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.256 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.256 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.256 \end{pmatrix} \tag{6.12}$$

Observation Model

The observation model receives measurements from the Ar.Drone. All measurements are treated as direct observation of the projected state variable. The measurements are received at a rate of 200Hz respectively every 5ms. Even the velocities, which are calculated based on image reorganization of the downward facing camera, are received every 5ms, although the camera has a frame rate of 60hz. This indicates that the measured data does not correspond to raw measurements, but is already post processed with an EKF within the quadcopter as explained in figure 6.2. This violates the assumption of normally distributed noise. The observation model receives the state variables as follows:

$$\mathbf{z} = \begin{pmatrix} z \\ \phi \\ \theta \\ \psi \\ v_x \\ v_y \end{pmatrix} \qquad \mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.13}$$

The output matrix $\mathbf{H}$ is an identity matrix. The measurement covariance noise matrix $\mathbf{R}_k$ is a $6x6$ diagonal matrix:

$$\mathbf{R}_k = \begin{pmatrix} 1e^{-2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e^{-5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1e^{-5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1e^{-5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1e^{-2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1e^{-2} \end{pmatrix} \tag{6.14}$$

The pitch and roll angles are measured very precise by the IMU. Thus, those values are trusted values and the elements in $\mathbf{R}_k$ have a small value. The measurements are 5 times more trusted than the model (compare elements of $\mathbf{Q}_k$ and $\mathbf{R}_k$). This leads to a behavior that the KF adjusts fast to the received measurements. The noise covariance values of the velocities are comparatively high due to the fact that

the independent noise assumption is highly violated.
The relative height z, and the yaw angle ψ are drifting over time. Thus, the entries of both values in the **H** matrix are set to zero as default.

6.3.2 Data synchronization

The measurements are transmitted from the Ar.Drone to ardrone_autonomy. All measurements are transmitted in one message. The package ardrone_autonomy converts the measurement into an odometry message. In case of the Ar.Drone the message consists of data from different sensors, namely, the orientation from the IMU and the velocities from the downward facing camera. Even though, the data is processed and published in one message the signals have different time delays due to the fact of different executing rates of the sensors and the post processing onboard the quadcopter. The KF receives the measurements from ardrone_autonomy. To incorporate delayed measurements in a KF is not trivial, especially while preserving the optimality of the filter. Therefore, the details of the data synchronization are explained in the further sections.

Time Delay Estimation

The total time delay of the closed loop varies and consists of several delays. The total delay is defined as time difference between the instant the state variable $\mathbf{x}_k$ is measured and the instant the control command, based on this measurement, is executed on the drone. The standard path of communication is: record a measurement, post process the measurement, transmit the data to the ground station, decode the data, calculate a new control command, transmit the control command back to the Ar.Drone, and finally execute the control command. The delay varies between 100ms and 250ms. Not all time delays are known or measurable. Thus, it is difficult to estimate the delays correctly. The total delay is separated

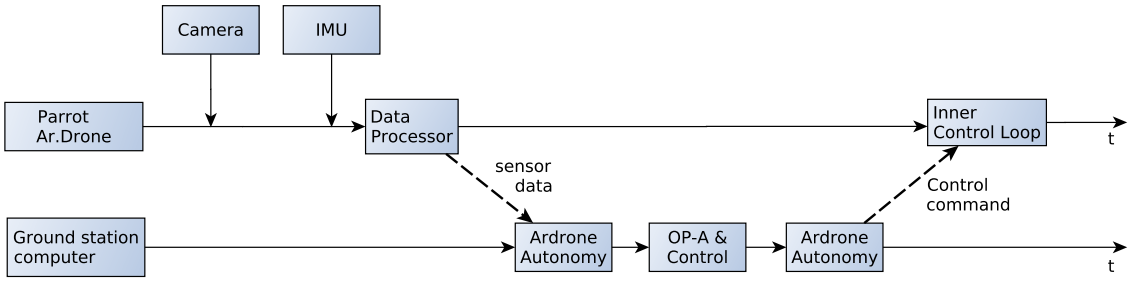


Figure 6.4: Sequence diagram of the processes that add a delay to the system.

into 4 components:

1. **IMU delay:** the IMU measures the orientation of the drone in space. During data acquisition the measurements are preprocessed and low-pass filtered. The researchers in [46] point out that a low-cost IMU, like the one used in the Ar.Drone, in comparison with professional IMU such as 3DM-GX2 will add an additional delay up to $\tau_{IMU} = 40$ms. Regardless to other measurements, the IMU has the smallest delay.

2. **Velocity delay:** The downward facing camera uses corner detection or optical flow to calculate the velocities of the quadcopter. Due to the huge computational costs the processing time of the velocities is significantly longer than the processing time of the IMU. The delay is defined as time span from the instant the image is recorded until it is internally available. The delay is unknown, but measurements based on a step input indicate that the average delay is present as: $\tau_{vel} =$

$\tau_{imu} + 20ms$. A second experiment based on the PTAM algorithm of the research group of the TU Munich [14] points out that the delay is $\tau_{vel} = \tau_{imu} + 40ms$. The algorithm uses the front facing camera to estimate a state vector. This vector is used as reference, because all state variables are subject to the same delay. Hence, the relative delay between the velocity and orientation was measured. However, it is not possible to verify how precise the results are. Both results are only a rough estimation. Furthermore, it is assumed that the delay is static.

3. **Transport delay:** The wireless LAN adds a non deterministic time delay to the system. Furthermore, it is unbounded and packet losses occur during the communication process over the network. The time delay varies strongly between $5-100$ms. The time delay is measurable by performing an echo request between the ground station and the Ar.Drone. The package size of 0.5kB equates the size of an odometry message or a control command. The echo request is executed every $500ms$. The time delay is calculated based on the last measurements to reduce the influence of a faulty measurement: $\tau_{LAN,k} = 0.7 \cdot \tau_{LAN,k} + 0.3 \cdot \tau_{LAN,k-1}$ [14].

4. **ground station:** From the instant an observation is received until a new control command is transmitted several processes on the ground station are involved. The processes are executed in a chain. Therefore, the total run-time of the chain is analyzed. It is assumed that the run-time is static. Measurements indicate that the total delay of the ground station is around $\tau_{com} = 20ms$.

Time Delay Measurement

The orientation, the velocity, and the rotor speed measurement delays are experimentally determined over several test flights by using the step response of the closed loop. For all three measurements we suppose the rotor dynamics are sufficiently fast and thus they do not have a delay. The mean over several test flights leads to the following results:

$$\begin{aligned} \tau_{meas,vel} &= 150ms \\ \tau_{meas,imu} &= 120ms \\ \tau_{meas,rot} &= 90ms \end{aligned} \tag{6.15}$$

Figure 6.5 and 6.6 visualize the delay of $\tau_{meas,vel}$ and $\tau_{meas,imu}$. It is important to note that during a stable connection the delay of the wireless LAN transport is continuously $20ms > \tau_{LAN}$. This confirms the assumption that the low-cost measurement unit as well as the data processing within the drone produce a huge delay. Figure 6.5 and 6.6 visualize the delay of $\tau_{meas,vel}$ and $\tau_{meas,imu}$. It is not possible to identify the processing time of the velocity τ_{vel} or the orientation τ_{imu} directly, instead the time span from the instant a control command was sent until the response was received and decoded at the ground station was measured:

$$\begin{aligned} \tau_{meas,vel} &= \tau_{vel} + \tau_{LAN} + \frac{\tau_{com}}{2} \\ \tau_{meas,imu} &= \tau_{imu} + \tau_{LAN} + \frac{\tau_{com}}{2} \\ \tau_{meas,rot} &= \tau_{rot} + \tau_{LAN} + \frac{\tau_{com}}{2} \end{aligned} \tag{6.16}$$

The delay consists of three parts: the processing time at the drone, the time to transmit the data from the Ar.Drone to the ground station and the other way around, and the computing time at the ground station. For the computing time at the ground station the half delay is assumed since no control commands are calculated and forwarded to the drone. The rotor speeds are analyzed because they are the first measurable parameters in the chain of measurements. A change of the rotor speeds induces a change in the orientation and this may induce translational velocities. Furthermore, the rotation speeds are not a subject to further processing. The rotor speed is measured with 200Hz and a small delay of $\tau_{rot} = 10ms$ is assumed to take the internal communication and processing of the Ar.Drone into account.

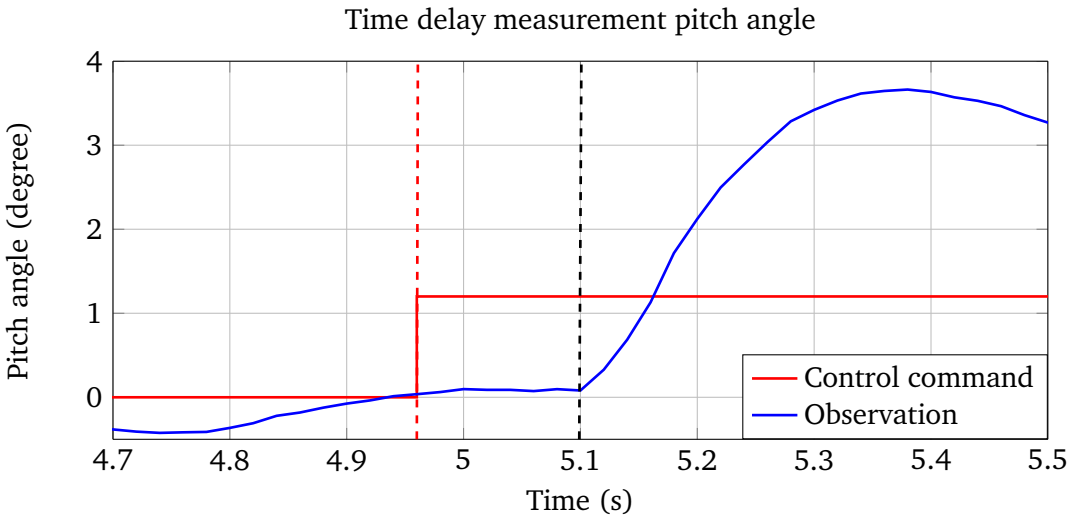


Figure 6.5: The total delay $\tau_{meas,imu} = 150ms$ is identified by manual aligning the progression line of control command to the instant a change in the pitch angle progression is visible. Here both instants are marked as red dashed line and as black dashed line. The total delay is defined as time span between both dashed lines. All orientation angles show similar characteristics.

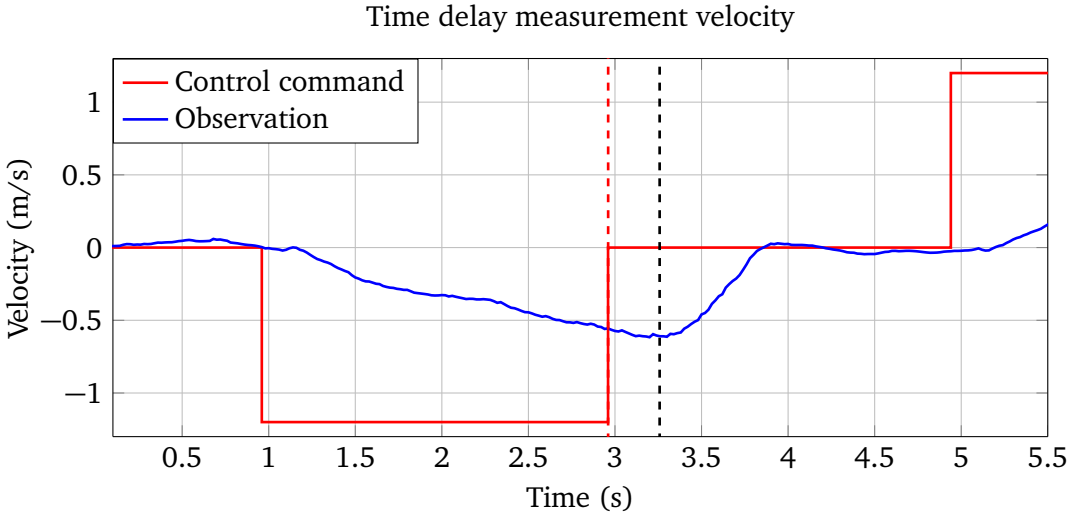


Figure 6.6: The figure presents the time difference between a control command is sent and the effect is visible at the ground station. The total delay of the velocity is calculated as the difference between the red dashed line and the black dashed line and is $\tau_{meas,vel} = 200ms$. The velocity in x- and y-direction have similar characteristics.

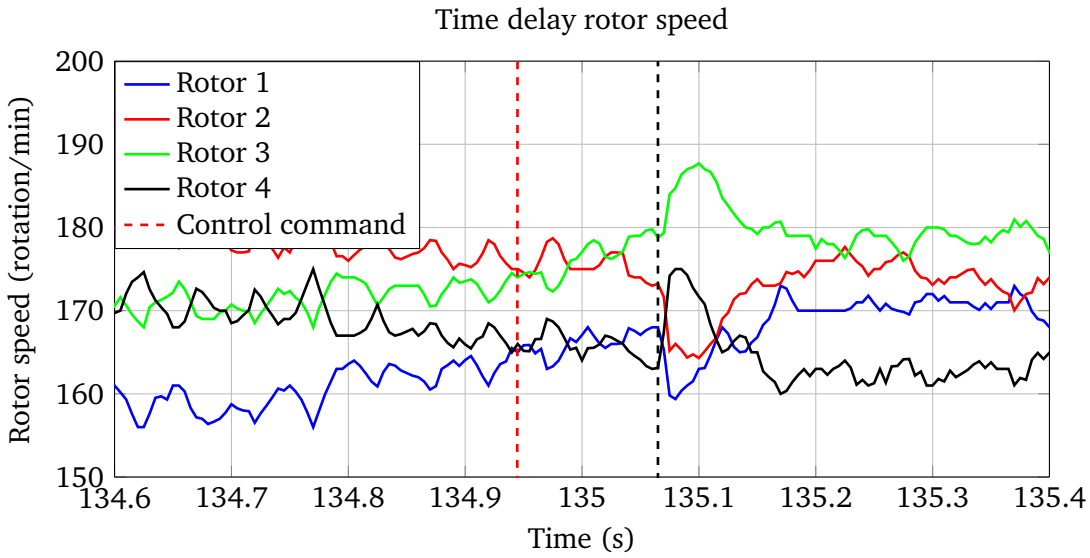


Figure 6.7: As soon as a control command is received by the Ar.Drone the inner control loop executes the command and adjusts the rotor speeds related to the new reference value. A control command is sent at 134.945s (red dashed line) and at 135.065s the rotor speeds start to change (black dashed line). The rotation speed of rotor 3 and rotor 4 is increasing with a high sloop. At the same moment the rotation speed of rotor 1 and rotor 2 are decreasing. The time difference between the black dashed line and the red dashed line indicates a delay of $\tau_{meas,rot} = 120ms$.

Time Delay KF

For the state estimation not all delays are required. Rather the time span between the orientation and velocity measurement is required. Either is the total delay of the system of interest to synchronize the control command with the received step response. This leads to the following three delays for the KF:

$$\begin{aligned} \tau_{KF,crt} &= \tau_{vel} + \tau_{LAN} + \tau_{com} \\ \tau_{KF,imu} &= 0ms \\ \tau_{KF,vel} &= 20ms \end{aligned} \tag{6.17}$$

The $\tau_{KF,imu}$ is zero because the orientation measurement arrives first. The velocity is available with a delay of $\tau_{KF,vel} = 20ms$. The delay $\tau_{KF,crt}$ is adjusted dynamically by using the echo request. The other both delays are static.

The aim of this section is to explain the path of the data in the KF, that is displayed in figure 6.8. As mentioned previously the measurements are time delayed and multiple-delays are present. Therefore, the data has to be synchronized to guarantee the optimality of the KF.

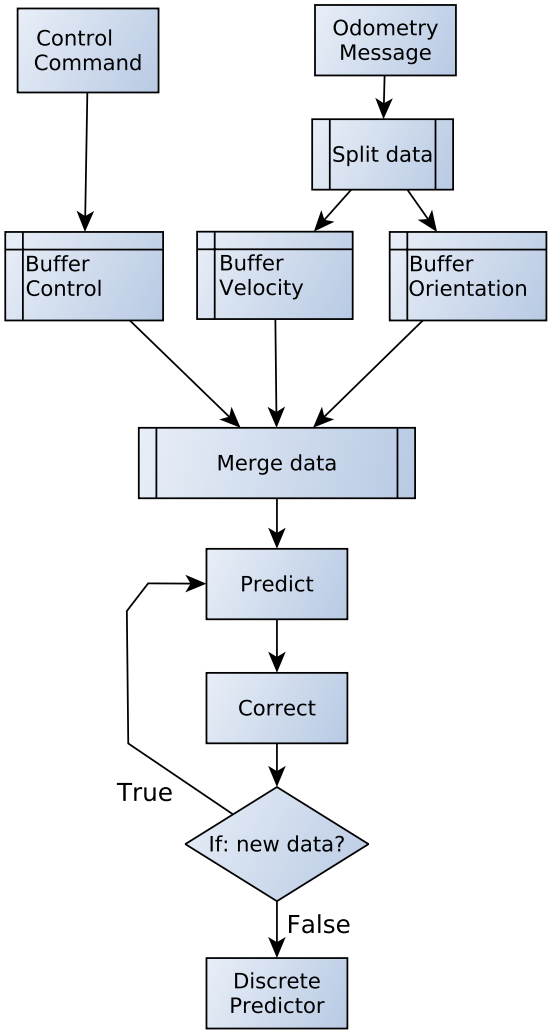


Figure 6.8: Flow chart: Implementation

The data are synchronized at the latest time stamp at which all data, in particular, the velocities, the orientation, and the control command are available. The data arrives in two packages. The first package is named odometry message and consists of the velocities and the orientation. Both have the same time stamp as they arrive in the same message. However, the orientation has less delay than the velocities. Hence, we assume that $\tau_{KF,imu} = 0ms$ and the velocities are recorded $\tau_{KF,vel} = 20ms$ earlier. The second message is the named control command. The KF requires the control command that fits to the received system response. Thus, the control term is exposed to the total time delay $\tau_{KF,crt}$ that is calculated dynamically.

To synchronize the data, we first untangle the data and store the velocities, orientation, and control command in separate buffers. The buffers order the data related to their time stamp (oldest message on the top). As next step we merge the different data to a new set of data synchronized in time. In this step, we access a start point in the buffer, depending on the delay, and move from that point on forward in time so that the delay is decreasing. In the next step, the data are handed as part of the main buffer to the KF.

The KF is executed at the same frame rate as the control commands are transmitted to the Ar.Drone, namely $100Hz$. Thus, every spin carries out two times prediction-correction cycle. However, the executing rate is flexible and be adjusted. The cycle is only carried out if all data are in stock. First a prediction of the time $\delta_t = \mathbf{z}_k - \mathbf{z}_{k-1}$ is performed. That is the time between the last and the actual measurement. However, the time is varying between $4-6ms$ and thus a flexible implementation based on the time stamps of the measurements is used, instead of using a fixed step size. Moreover, if δ_t is bigger than $10ms$ the prediction is split into time steps of $\delta_t = \mathbf{u}_k - \mathbf{u}_{k-1}$ that are equivalent to the time between two control commands. The predicted state vector $\mathbf{x}_{k+1}$ is handed to the correction process. This process compares predicted and corrected values and updates only the measured state variables $\mathbf{x}_k$. After the cycle is executed the used data is erased from the buffer. One exceptions are present, the buffer of the orientation is designed to store $\tau_{KF,vel}$ into the future of the execution time stamp, since the data is available earlier.

Yet, the data are not received every $5ms$. Instead the system shows non deterministic characteristics. Sometimes up to 8 messages arrive at the same time and afterwards a gap of $40ms$ arises where no data are received. To compensate such a behavior the algorithm only performs one cycle each spin and thereby does not process all data at once. If buffer are not empty but new measurements already arrive,

the old data is processed at once, to ensure the KF is always processing the latest available data. This is checked in the last step query of figure 6.8. A additional problem is that messages sometimes arrive with the same time stamp. Hence, it is not possible to identify which data carries the oldest measurement, because the data published by the Ar.Drone is not time stamped. Messages with the same time stamp lead to $\delta_t = 0$ and hence no prediction are executed, instead the data are directly handed over to the correction process.
In case of a bad connection no measurements are received. During all test flights phases of bad connection occurred. In average a bad connection will last up to 500ms. To guarantee a continuously available measurement, the prediction and correction cycle will run every 100ms even without measurements. In this case the predicted state vector $\mathbf{x}_{k+1}$ is directly forwarded to the output of the KF.
The output of the KF is an estimate of the state vector. This estimate is handed to a discrete predictor, to compensate the inherent time delays of the closed loop.

6.3.3 Discrete Predictor

The discrete predictor compensates the time delay. The system consists of two components:

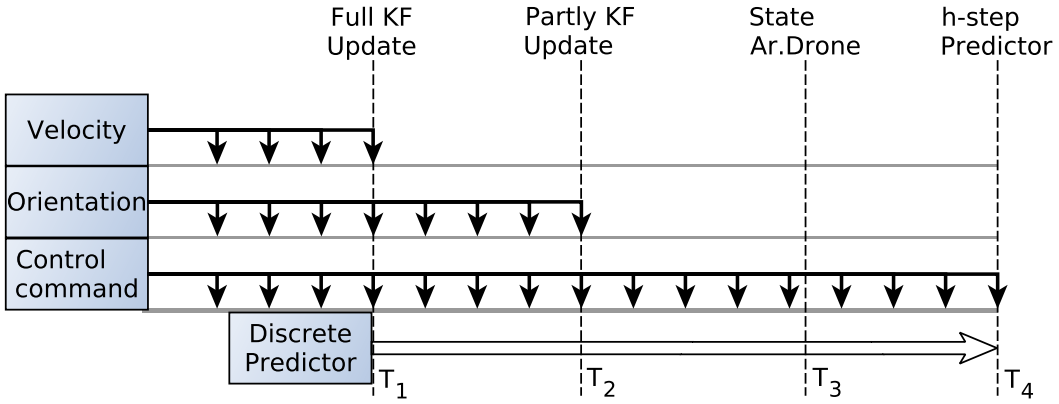


Figure 6.9: The sequence diagram displays the setup of the KF and the discrete predictor. The total setup consists of orientation and velocity observations, and the control command. At time T_1 the full KF is executed with all observations available. From that point forward the discrete predictor is active. Until T_2 the discrete predictor executes a partial KF with orientation observations. Afterwards, the h-step ahead predictor executes until T_4. The h-step ahead predictor only requires the control commands. At point T_1 a time delayed estimate of the state vector $\mathbf{x}_k$ is available and at point T_4 a estimate of the state vector with compensated time delay $\mathbf{x}_{k+h}$ is available. The time point T_3 is the actual state of the drone. The control commands between T_3 and T_4 have not yet been executed. Based on the future state vector $\mathbf{x}_{k+h}$ at time T_4 a new control command is calculated.

1. **Kalman Filter:** In the discrete predictor scheme a KF is used too, because the aim is to process all available observations. The main KF is only executed if all measurements are in stock (T_1). The last full cycle update is executed at T_1 equivalent to $t - \tau_{crt}$. The orientation data are 20ms further in time present. Hence, a partly execution of the prediction-correction cycle at time point T_2 is performed, only with the orientation observations available. This partial execution does not update the main state vector, but rather a new future state vector is created for the discrete predictor. In summary, two KF are executed simultaneously. By virtue of the second KF the time delay is be reduced by τ_{vel}.

2. **Predictor:** The remaining time delay is compensated by executing the h-steps ahead prediction scheme. During this time no measurements are available. The prediction is based on the last partly

updated KF state (T_2). The time steps of the prediction are $\delta_t = \mathbf{u}_k - \mathbf{u}_{k-1}$. The h-step ahead prediction scheme is identical with executing the prediction part of the KF in a loop. Thus, the new prediction is based on an old prediction. After every loop the used control term is erased from the control command buffer. The loop is executed until the buffer is empty, this is equivalent to the point in time at which the time delay has been fully compensated (T_4).

The discrete predictor is executed in every spin and based on the last state of the full update cycle. The predicted future state vector has no influence to the estimate of the main KF. The output of the discrete predictor is only an estimate of the state vector with compensated time delay. In case of a bad connection the predictor will be executed with the same strategic as the main KF. Hence, it will run every 100ms and jump from the partly KF directly to the h-step ahead predictor. Moreover, the prediction time is limited to 200ms, because we cannot guarantee a good approximation if we predict blind over a longer period of time.

6.3.4 Limitations robot localization

The package presents a good basic framework for the predictor based Kalman Filter. However, numerous limitations are present. Those limitations were tackled during the implementation and so the package was extended. The limitations are the following:

1. The model of motion is based on Newtonian dynamics instead of a quadcopter model.
2. The model of motion as well as the package itself do not allow to use control terms
3. The filter could not handle one or multiple time delays
4. No possibility to measure delays
5. All data were processed as soon as they were available
6. No discrete time predictor to compensate delays
7. The filter received delayed ROS messages because $tcp.NoDelay$ was not used

6.4 PID control

In order to control the translational velocities a nonlinear digital PID control scheme is implemented.

6.4.1 Velocity control

The velocity controller is a nested control loop. The inner loop is executed autonomously on the Ar.Drone [8]. The outer loop is executed at the ground station computer and calculates reference tilt angles for the inner control loop. The set ups are shown in figure 6.10. Both loops together form the velocity controller.

1. **Outer loop:** receives the future state vector $\mathbf{x}_{k+h}$. Based on those as well as the reference velocity $\mathbf{v}_{ref}$ the reference tilt angles θ_{ref},ϕ_{ref} for the inner control loop are computed. The outer control loop executes at a rate of 100Hz.
2. **Inner control loop:** consists of attitude control loop with a PI controller that calculates a reference angular rate based on the reference tilt angle calculated at the outer loop. And an angular rate control loop with a P controller that compares the reference angular rate with the measured angular rate. The inner control loop executes with 200Hz. The inner control loop is designed by Parrot and cannot be modified.

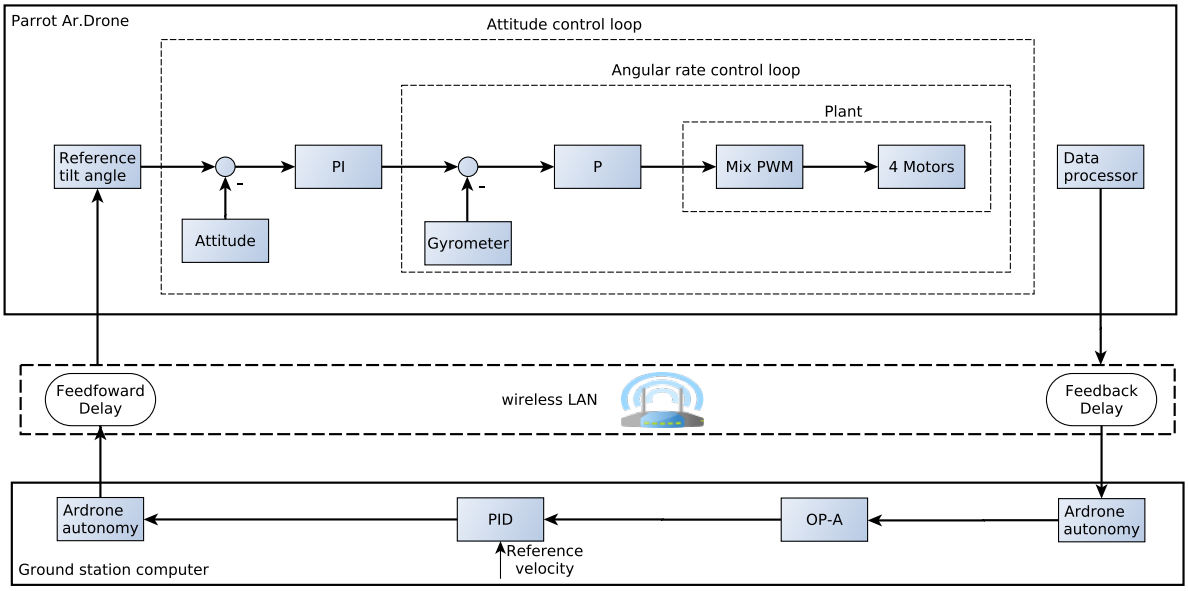


Figure 6.10: The setup shows the closed loop velocity controller. The controller is split into a inner control loop running on the Ar.Drone and a slower outer control loop that calculates reference tilt angles.

We use a PID controller with set point weighting, conditional integration, and a modified derivative term. Furthermore, we assume the identical dynamics for v_x and v_y and thereby use the same gains.

$$u = 0.3(0.9v_{k,ref} - v_k) + 0.07\sum e_k - 0.01\frac{v_k - v_{k-1}}{T} \tag{6.18}$$

In the derivative term the velocity signal instead of the error signal is used. This eliminates sudden changes in the derivative term. Furthermore, the sign of the derivative term is switched to counteract oscillations which may arise due to the badly compensated time delay in situations with a high delay. For the integral term we define an anti-windup value of $u_I = 0.7$ to limit the influence of the integral term. The derivative term is very sensitive to noise and sudden changes by virtue of the small sampling time. Hence, a low-pass filter is used for the derivative term but it introduces an additional time delay. The low pass filter is based on weighting the last three measurements and the last two outputs.

$$\begin{aligned} &\text{Cut-off frequency:} \quad \alpha = \tan(c_f 2\pi\frac{T}{2}) \qquad c = \frac{1}{\alpha} \\ &y = \frac{1}{1+c^2+1.414c}(u(z)z^{-2} + 2u(z)z^{-1} + u(z) - (c^2 - 1.414c + 1)y(z)^{-2} - (-2c^2+2)y(z)^{-1}) \end{aligned} \tag{6.19}$$

The Ar.Drone has no internal buffer and thus executes the last command until a new command arrives. In the situation of a bad wireless connection this may lead to an unpredictable behavior, because no new control commands are received by the drone. Based on those events crashes with the environment occurred. To minimize the risk of a crash during a bad connection a safety box is used. A bad connection starts with a sudden rise of the echo ping. If the ping reaches a threshold of $\tau_{tot} > 200ms$ a bad connection is assumed. In case of a bad connection a hover command is sent to the Ar.Drone and all maneuvers are instantly quit. The hover control works onboard the quadcopter and therefore it is not influenced by a bad connection. However, this approach is only based on the actual measurement of the ping instead of using a prediction of the ping.

7 Velocity Field Control (VFC)

In this chapter we introduce an application for the velocity controller framework of chapter 6. It is equivalent to a vector based control scheme and preferred in many application, instead of a position controller. A possible application can be coverage an tracking. This task may focuses on a cooperative scenario where an UAV supports a ground robot. To do so, the UAV has to track the ground robot, but while doing so the quadcopter should gather as much information as possible from the surrounding. Hence, the quadcopter has to fulfill two tasks simultaneously [53]. To dynamically calculate an optimal position to fulfill both tasks can lead to high computational costs. Therefore, a weighted sum of the reference vectors of both tasks may be used. The vector based approach reduces the computational costs. Moreover, the reference vector changes dynamically.
In this work we focus on a different application, namely the velocity field control. It is similar to potential field methods that are widely used in robotics. A potential field is designed by using an artificial potential function so that a global minimum is present. The gradient of the potential field points from any start towards the final point. We use a slightly different approach and directly design a gradient field referred to as velocity field. The velocity field as well as the potential field use a vector to control the quadcopter. The velocity field is used in conjunction with the proposed velocity controller to perform the task of path following [54, 55]. Furthermore, obstacle avoidance is integrated to ensure a collision free following of the path. A obstacle is designed as a Gaussian potential field which is then derived to obtain a velocity field. The theory is already applied successfully to kinematic ground robots. In this work we introduce the approach to quadcopters [13, 56].
A path is defined by a set of way-points in a world-frame. The task of path following is traditionally specified by approaching the way-points one after another based on a position control algorithm. However, this solutions are prone to disturbances and uncertainties of the system that may lead to bad performance [10]. The new approach specifies the desired path in a velocity field and uses the framework of chapter 6 to control the velocity, instead of a way-point based position controller. In section 7.1 the path following algorithm encodes the way-point into a time invariant velocity field. Every point in the world-frame is defined as reference vector. Due to the design of the velocity field the reference vector always points into the direction of the path. This helps to compensate uncertainties of the system and reduces the sensitivity against disturbances. Moreover, the velocity field control eliminates the need of timed trajectory to follow a path [11, 12, 57]. In section 7.2 the path following algorithm is expended to handle obstacle avoidance. An obstacle is defined as Gaussian potential field which then is derived to obtain a repulsive velocity field. The path following velocity field is overlain by the resulting obstacle velocity field to guarantee an absence of collision.
The approach is tested an in indoor environment. To realize the velocity field control it requires the pose of the quadcopter in the world-frame. Thus, an external mounted camera system is used to estimate the pose of the quadcopter in the world-frame. The pose estimation is carried out with a fiducial marker on top of the Ar.Drone. One major drawback of a direct visual control is the need for the Ar.Drone to stay in the field of view of the camera [11, 58].
The last section of this chapter describes the implementation of the proposed application as well as the integration of the camera system into the existing framework.

7.1 Path following

The task of path following is traditionally realized by successively approaching predefined way-points. The path in the example of figure 7.1 goes from point 0 to point 8. A path is encoded into a velocity

field based on three parameters: the position of the quadcopter in the world frame, the way-point of the shortest distance between the path and the quadcopter, and the next way-point on the path. The path following is performed in a two dimensional space in the x-y plane [59, 60].
A velocity field consists of two reference vectors. The first vector $\mathbf{v}_{tan}$ always points tangential to the path, and the second vector $\mathbf{v}_{approx}$ always points towards the nearest way-point on the path based on the actual position of the quadcopter. The reference velocity vector $\mathbf{v}_{ref}$ is a weighted sum of both vectors. The velocity vector has two degrees of freedom, namely: the length of the reference vector that is equivalent to the desired velocity and the orientation of the reference vector in space. Furthermore, it is possible to align the body-frame of the quadcopter with the way of the path by performing a rotation around the z-axis. In total, the algorithm has three degrees of freedom to perform the task of the path following.

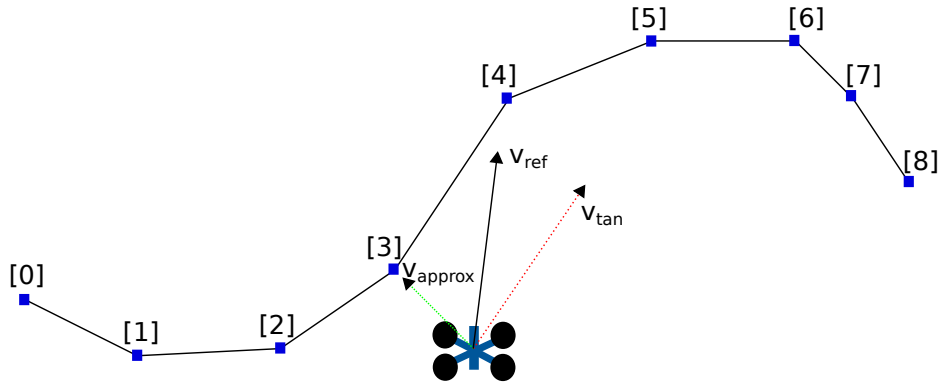


Figure 7.1: The path is defined by a set of 9 way-points. The direction of movements confirms with a rising number of the index. The velocity $\mathbf{v}_{approx}$ points towards the nearest way-point on the path. $\mathbf{v}_{tan}$ is defined as difference between way-point[4] and way-point[3]. $\mathbf{v}_{ref}$ is a weighted sum of both vectors.

Algorithm

Algorithm 1 Path following

1: **procedure** CALCULATE REFERENCE VELOCITY
2: $Index = Find\_nearest(Quad_{pos}, Path)$
3: *Velocities*:
4: $v_{tan} = Path[Index+1] - Path[Index]$
5: $V_{approx} = Path[Index] - Quad_{pos}$
6: *Constrains*:
7: $min\_dis\_full\_speed = \kappa$
8: $dis\_p\_near = \|v_{approx}\|$
9: *Clamping*:
10: **if** $v_{approx} > v_{max}$ **then return** $v_{approx} = v_{max}$
11: **if** $v_{approx} < -v_{max}$ **then return** $v_{approx} = -v_{max}$
12: **if** $dis\_p\_near < min\_dis\_full\_speed$ **then return** $v_{approx} = \frac{v_{approx}}{\|v_{approx}\|} \frac{dis\_p\_near}{min\_dis\_full\_speed}$
13: *Sum*:
14: **if** $dis\_p\_near < \nu$ **then return** $\alpha = \alpha_t$ and $\beta = \beta_t$
15: $v_{ref} = \alpha v_{approx} + \beta v_{tan}$

This paragraph explains the pseudo code of the path following algorithm step by step. The second line executes a function called $Find\_nearest$. It calculates the index of the way-point in the $Path$ array with

the shortest distance to the position of the quadcopter Quad_{pos}. The index is calculated in two steps. Firstly, the function calculates the difference between each point of the $Path$ array and the position of the quadcopter. This leads to an array of difference vectors. Secondly, the elements of the array are vectors and those are transformed into a distance using the Euclid norm. The $Index$ of the element with the smallest norm matches the $Index$ of the nearest point on the path (in example 7.1: way-point[3]). The next block $Velocities$ calculates the velocities $\mathbf{v}_{tan}$ and $\mathbf{v}_{approx}$:

$$\mathbf{v}_{tan} = \begin{pmatrix} \text{Path}_{near+1,x} \\ \text{Path}_{near+1,y} \end{pmatrix} - \begin{pmatrix} \text{Path}_{near,x} \\ \text{Path}_{near,y} \end{pmatrix}$$
$$\mathbf{v}_{approx} = \begin{pmatrix} \text{Path}_{near,x} \\ \text{Path}_{near,y} \end{pmatrix} - \begin{pmatrix} \text{Quad}_{pos,x} \\ \text{Quad}_{pos,y} \end{pmatrix} \tag{7.1}$$

The velocity $\mathbf{v}_{tan}$ is the difference between the next point on the path and the nearest point on the path (in example 7.1: way-point[4] - way-point[3]). The velocity $\mathbf{v}_{approx}$ is the difference between the nearest point on the path and the actual position of the quadcopter (in example 7.1: way-point[4] - Quad_{pos}). Both velocities are generated by the difference of two positions. The code block $Clamping$ limits the velocity of $\mathbf{v}_{approx}$ to ensure the validity of the linearization assumptions around the hover regime. The maximal flying speed is $\sqrt{\mathbf{v}_{approx}^2 + \mathbf{v}_{tan}^2}$. Furthermore, a border is defined to limit the zone of flying at maximum speed. If the quadcopter approaches towards the path and goes below the distance $min\_dis\_full\_speed = 0.5m$ then $\mathbf{v}_{approx}$ starts to decline towards the path. It is $v_{approx} = 0$ if the way-point on the path is reached. This reduces sudden changes in the velocity and creates a smooth approach towards the path. The distance between the path and the quadcopter is defined by dis_p_near. The last code block Sum calculates the reference velocity from the weighted sum of both velocities. The weighting factors are separated into two groups. The first set α_a and β_a are the weighting factors in the case that the quadcopter approaches the path. Thus, α_a is comparatively small to generate an almost tangential approach towards the path. The second set α_t and β_t are the weighting factors for the setting to hold the quadcopter on the path and therefore α_t is increased.
After the quadcopter reaches the path the main velocity is $\mathbf{v}_{tan}$ and points towards the next point on the path. The velocity $\mathbf{v}_{approx}$ is small and counteracts disturbances that push the quadcopter away from the track. Thus, the speed that defines how fast the quadcopter follows the path can easily be adjusted by changing the step size between two way-points, because $\mathbf{v}_{tan}$ directly depends on the distance between two points. In figure 7.1 the way-points [6]-[8] have a smaller distance between each other than the other way-points. This reduces the traveling speed at the curve and allows the Ar.Drone to follow the path more accurately.
If the quadcopter reaches the last way-point (in example 7.1: way-point[8]) the quadcopter should hover around this point. Thus, the $\mathbf{v}_{tan}$ is zero and $\mathbf{v}_{approx}$ points to the final point. This counteracts movements of the quadcopter to leave the final point.
The algorithm does not calculate a static velocity field for each point of the world-frame. Instead, it dynamically calculates only the desired reference velocity at the actual point of the quadcopter. At every execution of the algorithm a new reference velocity vector is computed.

Orientation

The quadcopter follows the path without changing the orientation. However, the quadcopter should always look in the direction of the path. To do so, the x-axis of the quadcopter has to be aligned with the path. The yaw angle rotates the quadcopter around the z-axis and thereby aligns the quadcopter with the path. Algorithm 2 shows the procedure to calculate the reference yaw angle. The yaw angle is calculated based on the scalar product of the unit vector in x-direction and the reference velocity vector. The scalar product is then dived through the norm of both vectors.

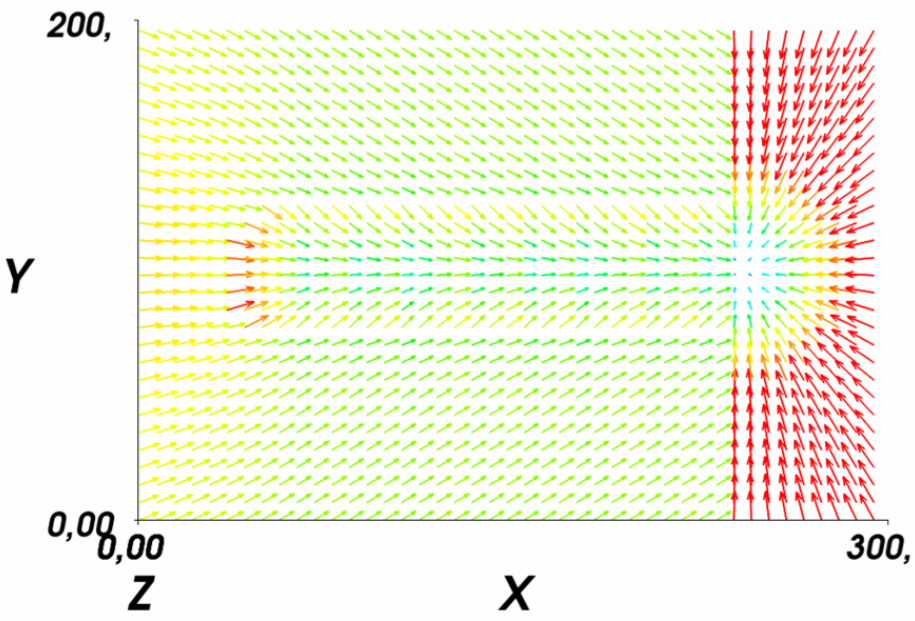


Figure 7.2: The figure shows a sample velocity field. The field is not displayed in world coordinates, instead it uses a map. The velocity field represents a straight line with a final point. If the quadcopter reaches the final point all reference vectors point towards the final point. Furthermore, The map shows impressively the impact of the weighting factors such that a clear border is visible in the middle of the map that separates the two areas, namely approaching and tracking.

Algorithm 2 Path following

1: **procedure** CALCULATE REFERENCE YAW ANGLE
2: $a = [1;0]$
3: $b = [v_{ref,x}; v_{ref,y}]$
4: $cos = \frac{a * b}{\|a\| \, \|b\|}$

7.2 Obstacle Avoidance

Obstacles are represented as a Gaussian Potential field which then is derived to obtain a vector field. The reference velocity vector points away from the object to avoid a collision. Objects are designed as a multivariate normal distribution such as a 2D Gaussian distribution. The general design of the distribution is as follows:

$$f(x,y) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-p^2}} \exp[\frac{1}{2(1-p^2)}[\frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2} - 2\frac{p(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y)}]] \tag{7.2}$$

The distribution is defined by five parameters. The parameter μ_x and μ_y set the origin of the obstacle. The variance σ_x and σ_y define the width and the Gaussian distribution. If σ is small the most values are in a small band of the Gaussian curve and thus the curve has a steep slope. Here, symmetric distributions are used so that $\sigma_x = \sigma_y$. The parameter p describes the correlation between x and y and is $p = 0$. To adapt the general design of a 2D Gaussian distribution to our application we define a general weighting term:

$$A = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-p^2}} \tag{7.3}$$

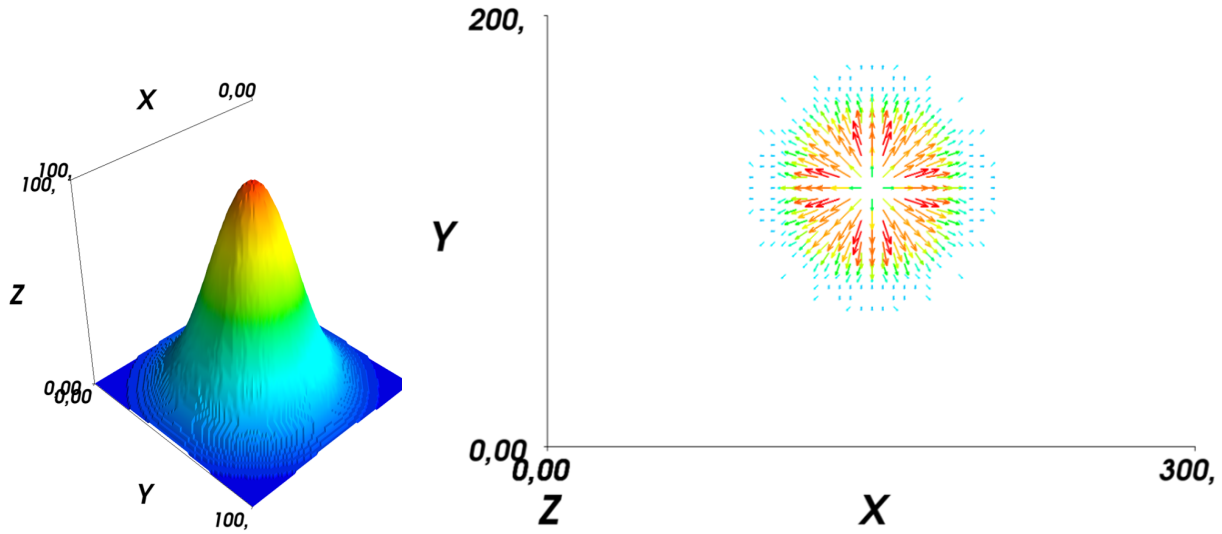
Furthermore, σ is the main tuning parameter, because it directly correlates with the size of the velocity field. The parameter is defined as follows:

$$\sigma = \frac{size\_obstacle + size\_quadcopter}{2\gamma} \tag{7.4}$$

The velocity field should not only be active if the quadcopter is very close to the obstacle. Therefore, the size of the obstacle is virtually expended by the size of the quadcopter. As result the new size is defined as sum of both objects, instead of using the size of the obstacle alone. The term is divided by two, because the field is defined from the center of the object. In addition a weighting factor γ is used for scaling.
The Gaussian distribution itself is not a velocity vector. Hence, the gradient of the distribution is a velocity field (the gradient of the distribution in the left figure figure of 7.3 is shown in the right figure of 7.3). The slope of the distribution directly correlates with the velcoity.

$$v_{ref} = \text{grad} f(x_{quad}, y_{quad}) = \begin{pmatrix} \frac{\delta f}{\delta x} \\ \frac{\delta f}{\delta y} \end{pmatrix} \tag{7.5}$$

The velocity field for the obstacle avoidance is static and superpose the path following velocity field at



(a) 2D Gaussian Distribution (b) Repulsive velocity field

Figure 7.3: The left figures shows the Gaussian distribution. The right figure shows gradient of the 2D Gaussian curve creates a velocity field that points away from the center of the object. The velocity decreases when going away from the center of the obstacle.

the point of the quadcopter. The new reference velocity is a weighted sum of the reference velocity from path following and the reference velocity from obstacle avoidance:

$$v_{ref} = \frac{\xi v_{ref,path} + \chi v_{ref,obst}}{2} \tag{7.6}$$

The parameter ξ and χ add two degrees of freedom to adjust the sum of both reference velocities.

7.3 Implementation

This section explains how to implement the path following and the obstacle avoidance algorithm in the existent framework of chapter 6.
The application of path following needs the position of the quadcopter in the world-frame to calculate a reference velocity and orientation. Unfortunately, only relative state variables in the body-frame such as velocity are available from the internal measurements of the Ar.Drone, instead of the position in the world-frame. To solve this a problem an external camera, fixed on the ceiling of the laboratory, is used to estimate the pose of the quadcopter in space. To do so, an aruco marker is fixed on the top of the quadcopter. Thus, an image recognition algorithm estimates the pose of the quadcopter in the world-frame.
The section is organized as follows: firstly, section 7.3.1 presents an overview of the approach. Afterwards, section 7.3.3 describes how the ground truth pose is estimated and how the a ground truth velocity is calculated. Section 7.3.4 parametrizes the path following and obstacle avoidance algorithm. In section 7.3.5 a yaw controller is implemented based on the ground truth pose estimation of the camera system to align the quadcopter with the path.

7.3.1 Approach

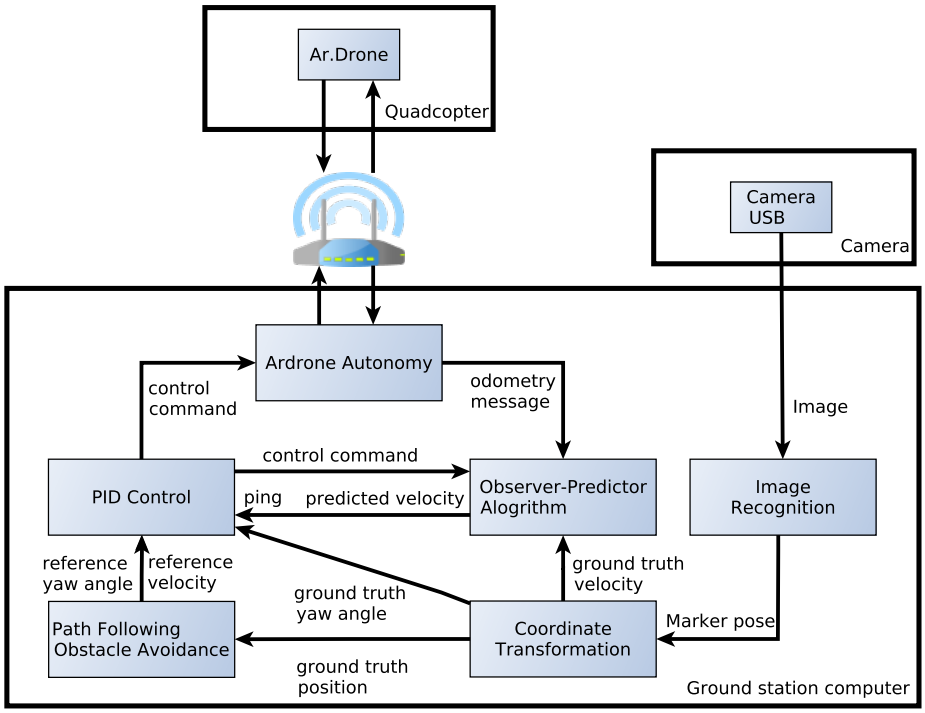


Figure 7.4: The illustration shows the structure of the modified approach. As application path following and obstacle avoidance are used. Moreover, the system is expended of three processes, namely camera USB, image recognition, and coordinate transformation. The new processes are used to provide a ground truth estimate of the pose and velocity of the quadcopter.

Here, we expend the framework developed in chapter 6 to perform the task of path following. Therefore, the processes at the ground station are modified. The data handling between ground station and quadcopter as well as the processes on the Ar.Drone are identical to the in chapter 6 introduced framework. The modified approach is shown in figure 7.4 and consists of the following processes:

1. **Camera USB:** an external camera, mounted on the ceiling of the laboratroy, overviews the whole flying space. The camera sends the image to the ground station computer via a USB connection [61].

2. **Image Recognition:** The camera is used for visual servoing. To do so, an aruco marker is fixed on the Ar.Drone. The received image is processed and a marker detection algorithm is executed to estimate the pose of the marker. The main drawback is that the quadcopter, or at least the marker, has to be in the field of view of the camera [62, 63, 58].

3. **Coordinate Transformation:** the process performs a nonlinear coordinate transformation to compute the position, orientation, and velocity of the quadcopter. The position, and orientation are represented in the world-frame. The velocity is transformed into the body-frame of the quadcopter.

4. **OP-A:** the ground truth velocities are sent to the OP-A. The KF of the OP-A carries out a data fusion between the velocity estimate of the Ar.Drone itself and the ground truth velocity estimate of the coordinate transformation process. Both velocity estimates are merged to optimize the estimation. However, they have different update rates. The ground truth velocity is calculated in the process before to ensure the linear approach of the KF. Thus, the no nonlinear calculations are performed in the OP-A. This helps to reduce the computational workload of the system. The ground truth velocity is roughly available every 10th spin. So that in the interim the velocity is estimated only with the data available from the Ar.Drone.

5. **Path Following and Obstacle Avoidance:** is the application of the framework. It receives messages from the ground truth position and yaw angle of the quadcopter. Both values are required to compute the reference velocity and yaw angle of the quadcopter to follow the path. The reference values are transmitted to the PID control scheme.

6. **PID Control:** The PID control scheme calculates the control command for the Ar.Drone and consists of two controllers. First, the velocity controller introduced in chapter 6. Second, a P-controller to control the yaw angle of the Ar.Drone. Both commands are calculated simultaneously and send as one message to the quadcopter .

7.3.2 Image recognition

This paragraph illustrates the procedure of image recognition and describes applied packages:

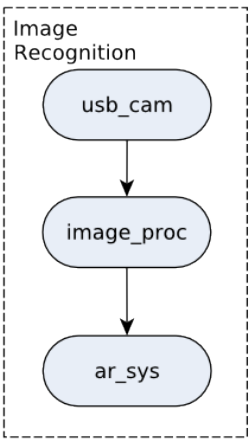
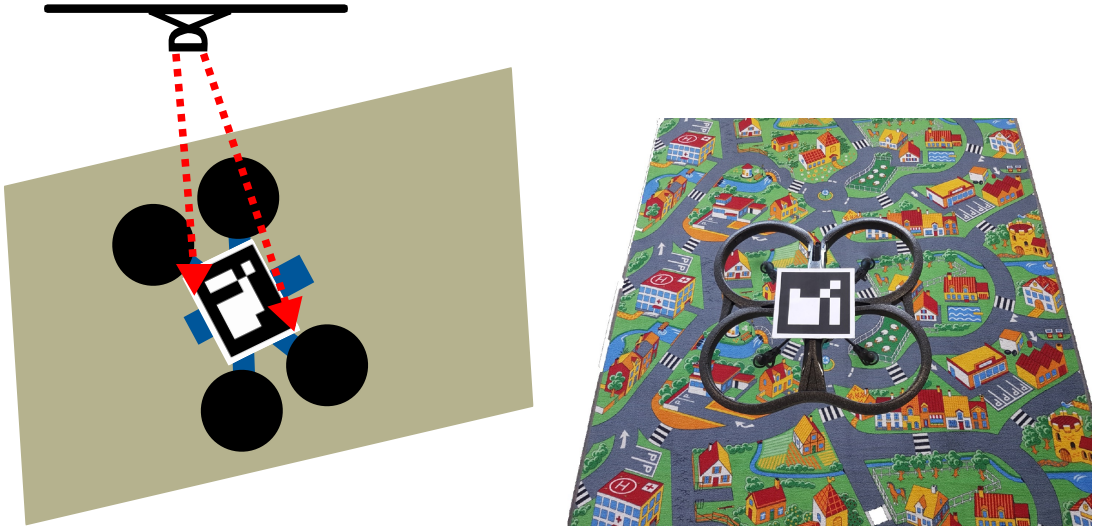


Figure 7.5

1. **usb_cam:** the first package is developed by Bosch and contains a ROS driver for USB cameras. It grabs the image from the USB interface and publishes the image as a ROS topic.
   The camera captures images at a resolution of $1980x1080$ pixel to use the maximum available field of view. Using such a high resolution leads to a low frame rate of roughly $15hz$. The exposure time is decreased by manual setting to reduce the blur effects. However, this requires a bright environment. The auto focus is manually adjusted, too. The focus value is set at a low value (focus=17) to ensure an optimal image quality at the operational height of the quadcopter. The settings are directly sent to the camera using the package $qv4l2$ [61].

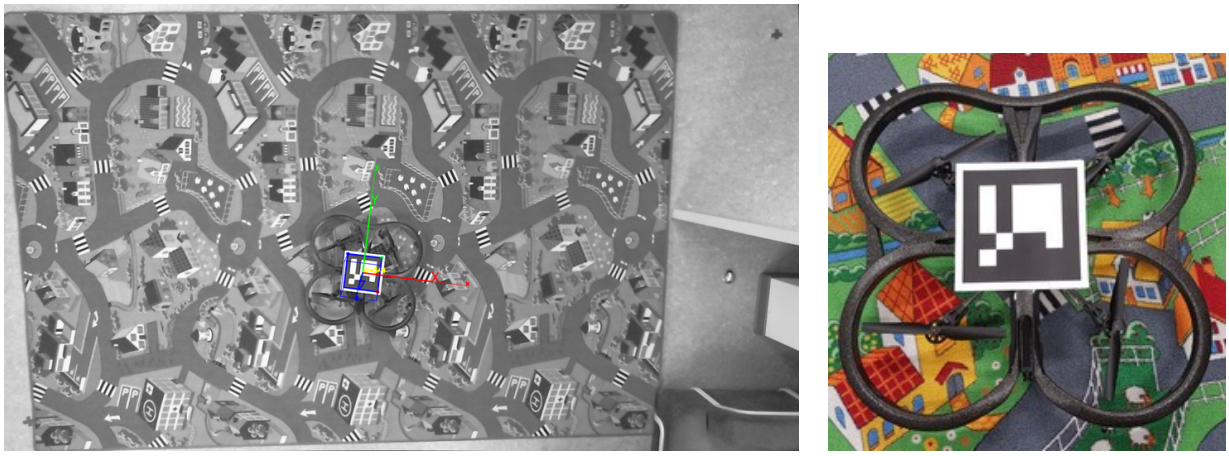
2. **image_proc:** this package grabs from the published image and processes it. The process removes the camera distortion and thus rectifies the image [62].

3. **ar_sys:** detects aruco markers in the rectified camera image and estimates the pose of the marker. The distance between camera and marker varies between $2 - 2.5m$. Thus, the marker size should be not smaller than a quadratic edge length of $0.16m$. In addition a bounding of $0.01m$ is put around the marker to separate the marker from the background. The marker detection works very precise, with a variance of less than $0.01m$ in a static condition. The processing time of the ar_sys algorithm is around $60ms$. This adds an additional delay to the system [63].



(a) Setup (b) Laboratory

Figure 7.6: The illustration shows the experimental setup. The quadcopter flies on a carpet. In addition it has an aruco marker fixed on top of it. The ceiling camera of the laboratory takes a video stream and sends it the ground station computer.



(a) Result marker detection (b) Aruco marker

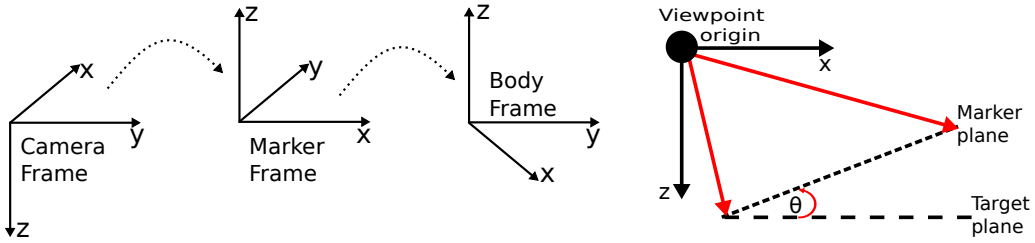
Figure 7.7: Figure (a) shows the result of the ar_sys package in a black/white image. The pose of the marker is correctly estimated and a new coordinate frame is created in the center of the marker. Figure (b) shows that the aruco marker is attached in the center of the quadcopter.

To ensure an optimal estimation of the ground truth velocity, the camera system was calibrated with a checkerboard and resulted in a linear squared line error of $LSE = 0.11$. Furthermore, the ground truth velocity estimation was evaluated with a ground robot. Therefore, a predefined track of the size of 1m was used in combination with a video stream. The video was analyzed to manually measure the time

span the ground robot needs to move $1m$. Based on that value the average velocity was calculated. This reference value was compared with the ground truth velocity estimation. The test was repeated several times and the average error was less than 2.25%.

7.3.3 Ground Truth Position and Velocity Estimation

This paragraph describes the coordinate transformations. The process receives a pose estimation of the marker in the camera-frame that is equal to the world-frame. It executes nonlinear coordinate transformations to align the marker-frame with the body-frame of the Ar.Drone. The different coordinate frames are shown in figure 7.8. It is important to note that the body-frame uses a coordinate frame according to REP-103, instead of using the body-frame defined by Parrot (rotation around x-axis of 180°). The transformation from camera-frame to body-frame is split into two transformations with a marker-frame in the middle. The marker frame y-direction is aligned with the negative x-direction of the quadcopter. Thus, the x-direction of the marker-frame is aligned with the negative y-direction of the body-frame. This leads to the following transformations that are based on equation 4.1. The first transformation is $\mathbf{R}(180°,0,90°)$. The second transformation is $\mathbf{R}(0,0,90°)$. The transformations between marker- and body-frame are static. However, the quadcopter rotates out of the x-y plane during



(a) Process of coordinate transformations.

(b) Project marker into plain target plane Roll

Figure 7.8: Figure (a) shows both nonlinear coordinate transformations that are executed to transform a measurement from the camera-frame to the body-frame of the Ar.Drone. Figure (b) shows that the marker is outside of the x-y plane. Hence, the marker plane is projected into a plain target plane in the x-y plane.

a flight due to the fact that movements are based on rotations around the x- and y-axis. For this reason the marker moves out of the x-y plane, too. To get a correct estimate of the position the marker is projected into a plain target plane in the x-y plane (compare figure 7.8). The pitch angle and roll angle rotations are inverted to get a correct pose of the quadcopter in the target plane of figure 7.8. Moreover, a translational displacement between the marker-frame and body-frame exists. This is simply counteracted by a translation offset of $\mathbf{x}_o = [0.03,0.0,0.07]$.

Velocity Estimation

The velocity is computed as derivative of the position estimate in the world-frame. This leads to the velocity:

$$\mathbf{v}_{cam} = \frac{\mathbf{x}_{k,cam} - \mathbf{x}_{k-1,cam}}{T_k} \tag{7.7}$$

The position is given through $\mathbf{x}_{k,cam}$ and the time step size is T_k. The position is estimated with a time-varying frame rate of roughly $15Hz$. The velocity is very sensible to noise. The use of the full frame rate leads to a bad velocity estimate, because the change of position between two points is within the noise level of the signal. Two approaches are possible to counteract the issue. First, a low pass filter can be used. However, this adds an additional delay, since the output is based on old inputs. Second, a reduction of the frame rate lifts the position changes between two measurements out of the noise level.

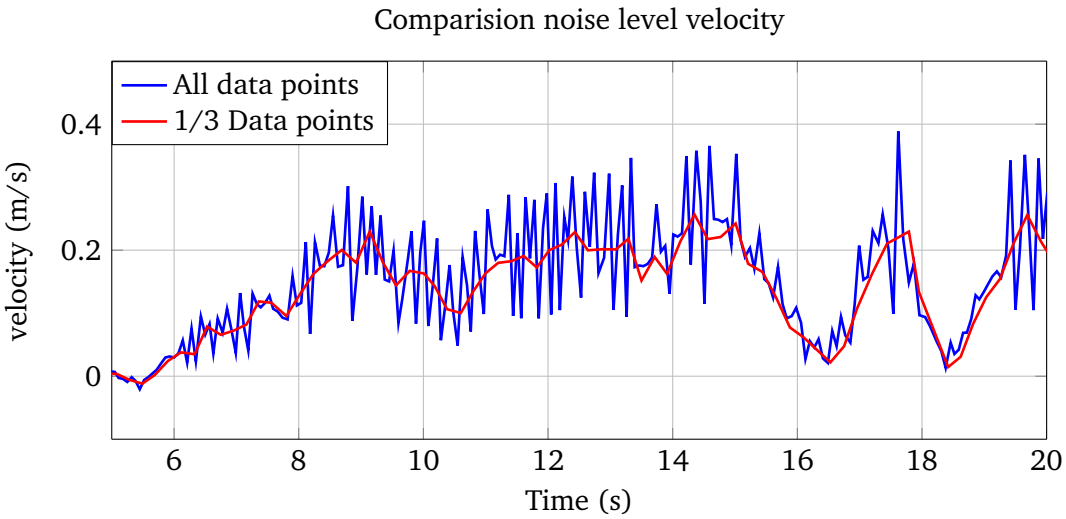


Figure 7.9: The figure shows the ground truth velocity based on the pose estimation of the external camera system. At full frame rate the velocity estimate is overlain by noise. A reduction of the frame to every second measurement reduces the noise dramatically.

We use the second approach and reduce the frame rate for velocity calculation to $5Hz$ such that every second position estimate is used:

$$\mathbf{v}_{cam} = \frac{\mathbf{x}_{k,cam} - \mathbf{x}_{k-3,cam}}{T} \qquad T = T_k - T_{k-3} \tag{7.8}$$

Next the velocity is transformed into the body-frame of the Ar.Drone. To do so, the rotation matrix of equation 4.1 is used. This leads to the following:

$$\mathbf{R}(\pi,0,\psi) := \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ \sin\psi & -\cos\psi & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{7.9}$$

Equation 7.8 and 7.9 lead to the ground truth velocity of the Ar.Drone:

$$\begin{aligned} v_{x,body} &= v_{x,cam}\cos\psi \; + \; v_{y,cam}\sin\psi \\ v_{y,body} &= v_{x,cam}\sin\psi \; - \; v_{y,cam}\cos\psi \end{aligned} \tag{7.10}$$

7.3.4 Parametrization Path Following and Obstacle Avoidance

This section briefly explains the parameters in path following and obstacle avoidance algorithms. The parameters were determined experimentally and proofed to be good trade off between the different tasks.

Path Following

The algorithm has a set of variables that are carefully chosen. The first group is $min\_dis\_full\_speed$ and v_{max}. Both variables constrain the maximum flying speed of the quadcopter. Here, we choose $min\_dis\_full\_speed = 0.5$ and v_{max} is limited to $0.4m/s$. If the quadcopter approaches from a distance farther than $0.5m$ it is approaching with full speed $v_{approx} = 0.3m/s$ until it reaches the border of $0.5m$.

Variable	Approach	Path
α	0.3	0.5
β	2.0	2.0

Table 7.1: Parameter path following

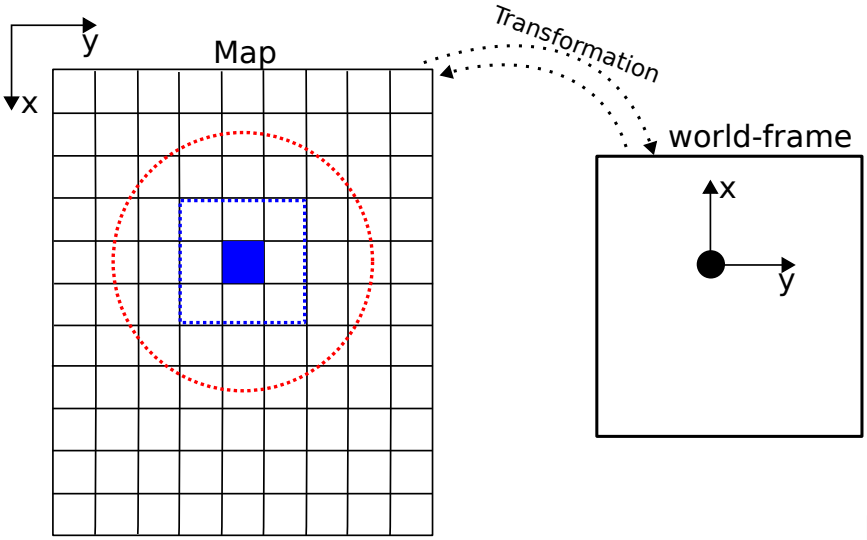
From that point onwards the velocity v_{approx} decreases in the direction to the path.
The parameter α and β are determined experimentally, too. The values are displayed in table 7.1. The distance between the path and the quadcopter trigger the switch of variables. If the quadcopter is less than $dis\_p\_near < 0.3m$ to the track the parameter set switches to path. This allows the system to counteract small disturbances more effectively.

Obstacle avoidance

The obstacle avoidance algorithm is precomputed and overlain by the path following reference velocity. An obstacle is predefined in a map. Then all obstacles can be directly added to the map, or only the obstacle contribution next to the position of the quadcopter is considered and thus only the reference velocity in one point is calculated. The map consists of a matrix mesh grid with $300x200$ squares. Each square represents a size of $1x1cm$. This results in a reasonable precise velocity field. Moreover, a precise grid is required to model the Gaussian curve precise. The velocity is computed based on the gradient of the curve. Due to the finite modeling of the curve, if the amount of points is too small, the algorithm models the Gaussian curve with partly plain surfaces. The same happens if the slope is too small. Plain surfaces lead to a gradient of zero and therefore they should be eliminated. This is reached by increasing the amount of points as well as using a high slope of $A = 300$. Due to the fact that the slope is high, the gradient produces high velocities $v_{ref,obst} > 1m/s$. However, such high velocities would eliminate the balance of the system. Therefore, $v_{ref,obst}$ is multiplied by a factor $\iota = \frac{1}{12}$ and clamped at velocity maximum of $v_{ref,obst} = 0.4m/s$.
The procedure is split into four parts. First, transform the position of the quadcopter into the map. Second, get the reference velocity for obstacle avoidance. Third, transfer the map velocity into the world-frame, and fourth superpose both reference velocities. Step three is needed because the map coordinate frame and the body coordinate frame have a different axis orientations. Moreover, step one and three are required due to the fact that both coordinate frames have different origins. The origin of the world-frame is the center of the camera and the origin of the map is the top left corner.

7.3.5 Yaw angle control

The yaw angle control has a similar structure as the velocity control. However, the setup is simpler. The Ar.Drone provides no drift free yaw angle measurement. Hence, the drift free yaw angle estimate from marker detection is used. The outer loop calculates the reference angular rate that is sent to the Ar.Drone through a proportional term. The inner loop consists of one loop less than the velocity controller and thereby directly receives an angular rate set point from the ground station. The inner loop is a proportional controller [1, 2].



[h!]

Figure 7.10: The left part of the figure shows an obstacle in the map configuration. Each square represents a point on the map for a which a reference velocity is computed. The blue square is the obstacle itself, the blue dashed square represents a virtual expansion of the object, and the red dashed circle shows the area in which the velocity field is active. Furthermore, it shows that the velocity field is only active near to the object. The reference velocity is equal to zero for points far away from the object. First, the world-frame position is transformed into the map-coordinates. Then, the reference velocity is computed. Finally, the reference velocity is transformed back into the world-frame.

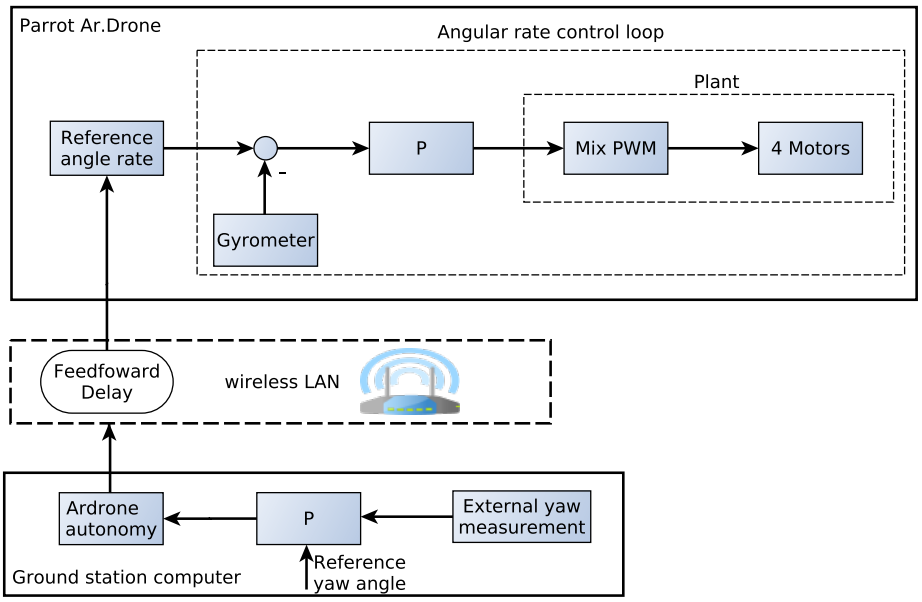


Figure 7.11: The setup shows the closed loop yaw angle controller. The controller is split into a inner control loop onboard the Ar.Drone and a slower outer control loop that calculates reference yaw angular rate.

The controller is defined as:

$$u = 0.4(\psi_{k,ref} - \psi_k) \tag{7.11}$$

The yaw controller is integrated into the framework as follows: first the quadcopter is aligned towards the reference path. If the difference between ψ_{ref} and ψ is bigger than 15° than the quadcopter first rotates towards the path until the error is smaller than $e < 5°$. If the error is under the threshold the quadcopter starts to follow the path. However, if the error rises again over $e > 15°$ and the quadcopter

is not on track than the reference velocity is set to zero and the procedure starts from the beginning. Thus, in total three state variables are controlled simultaneously.

8 Results

This chapter evaluates the performance of the proposed OP-A, the velocity controller, and the path following application. The overall performance is based on a chain of processes. First of all, a good estimate of the future state $\mathbf{x}_{k+h}$ is required. This results are forwarded to the velocity controller. Finally, the results of the path following application highly depends on the performance of the velocity controller. The results are based on data that were obtained through numerous of test flights with the Parrot Ar.Drone 2.0.
In the first section of this chapter, section 8.1, the time delay compensation and the quality of the state vector estimation $\mathbf{x}_k$ are evaluated. The second section analyses the capabilities and limitations of the proposed velocity controller. The last section, focuses on path following and obstacle avoidance application. Furthermore, it illustrates the possibilities and challenges of using a velocity field based control. For all obtained data the following assumptions are valid:

1. **Indoor:** all experiments were carried out in the laboratory. Furthermore, a highly textured carpet was used to ensure the best possible velocity estimation.

2. **External data:** The sections time delay compensation and velocity controller do not use any external measurements such as ground truth velocities or the absolute position in space. However, the sections path following and obstacle avoidance use an external camera system to estimate the pose of the quadcopter.

3. **OP-A and velocity controller:** Both processes are executed at a frame rate of $100Hz$.

4. **Horizontal speed:** the maximum speed in indoor environments is limited to $0.4m/s$.

8.1 Time Delay Compensation

A main part of this work was the analyses and compensation of the present time delays in the closed loop. The measurements arrive asynchronous such that up to 8 messages are received at the same time with the same time stamp, or for gaps up to $40ms$ no measurements arrive at all. In addition, the observation of each sensor has different time delays. Therefore, it was of great relevance to synchronize the received data. This was done in the Kalman Filter. The filter provides an estimate of the state vector $\mathbf{x}_k$. Afterwards the estimated state vector is forwarded to the discrete predictor scheme to compensate the time delays. The result is the future state vector $\mathbf{x}_{k+h}$. Both tasks go hand in hand and are carried out in the OP-A.
An accurate estimate of the future state vector $\mathbf{x}_{k+h}$ is required to guarantee a good performance of the PID controller. Therefore, we focus in this section of the performance of the discrete predictor. First, the evaluation methods are explained:

Qualitative Validation

To present qualitative results of this work, different scenarios are presented in the following figures: 8.1,8.2,8.3,8.4. A figure usually contains three graphs. The future state vector $\mathbf{x}_{k+h}$ at time T_4 is drawn in red, the filtered and synchronized estimate of the state vector $\mathbf{x}_k$ at time T_1 is drawn in green, and the measurement from the Ar.Drone $\mathbf{z}_k$ at time T_1 is drawn in blue. The inherent delay is clearly visible in all figures.

Quantitative Evaluation

In Addition, a quantitative evaluation was carried out. To do this, the prediction is compared to the measurement shifted in time to align both progressions. Furthermore, the filtered data are compared with the unfiltered measurement, since both are delayed the same amount of time, no shifts are needed. To evaluate the data the root mean squared error between:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{T_1}(t_i + t_d) - x_{T_4}(t_i))^2} \tag{8.1}$$

is used.

Results

The element in the chain is the KF. The results of the Kalman Filter are tuned by adjusting the covariance values of the process and measurement noise, and the prediction model. The prediction model has only a little influence at the full KF update at time T_1. Besides of data synchronization, the KF is mainly smoothing the measurements $\mathbf{z}_k$ from the Ar.Drone. The figures 8.1 and 8.2 illustrate that the

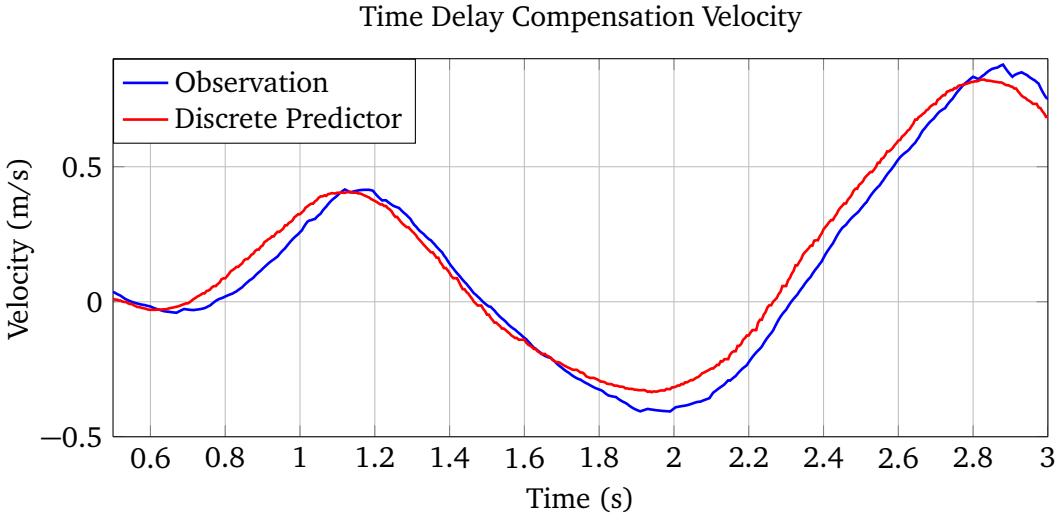


Figure 8.1: The figure shows the observation $\mathbf{z}_k$ from the Ar.Drone (blue line), and the predicted future state vector $\mathbf{x}_{k+h}$ (red line). Time delay is $t_d = 0.120s$.

discrete predictor successfully counteracts the inherent delays in the closed loop. In particular, figure 8.1 indicates that the future state $\mathbf{x}_{k+h}$ velocity estimate, output of the discrete predictor, has a lead in time of roughly $t_d = 0.120s$ in comparison to the time delayed estimate $\mathbf{x}_k$. At time $t = 4.3s$ a step input is sent to the quadcopter. A comparison between $\mathbf{x}_{k+h}$ and the aligned observation $\mathbf{z}_{k+h}$ indicates the prediction matches the measurement very well. The accuracy of the prediction fluctuates depending on external circumstances. For instance, very fast dynamics lead to a worse prediction performance, due to the fact that the system is modeled for dynamics around the hover regime. A poor prediction performance is present if the quadcopter is in hover mode, because in hover mode the onboard controller is active and no control commands are transmitted to the quadcopter and the actuated control commands of the onboard controller are not sent to the ground station. Thus, the prediction is only based on the last observations $\mathbf{z}_k$.

Figure 8.2 reveals that the orientation estimate of $\mathbf{x}_k$ is overlain by the orientation observations $\mathbf{z}_k$. This means that the tilt angle measurements are assumed as correct. The future state prediction $\mathbf{x}_{k+h}$ of the orientation is analyzed. The prediction carried out to T_4 leads to worse results than the velocity prediction, because the prediction model is based on the identified model of chapter 4.2. The figure 8.2 identifies that the predicted estimate is not able to predict the peaks of the tilt angle precisely. It seems

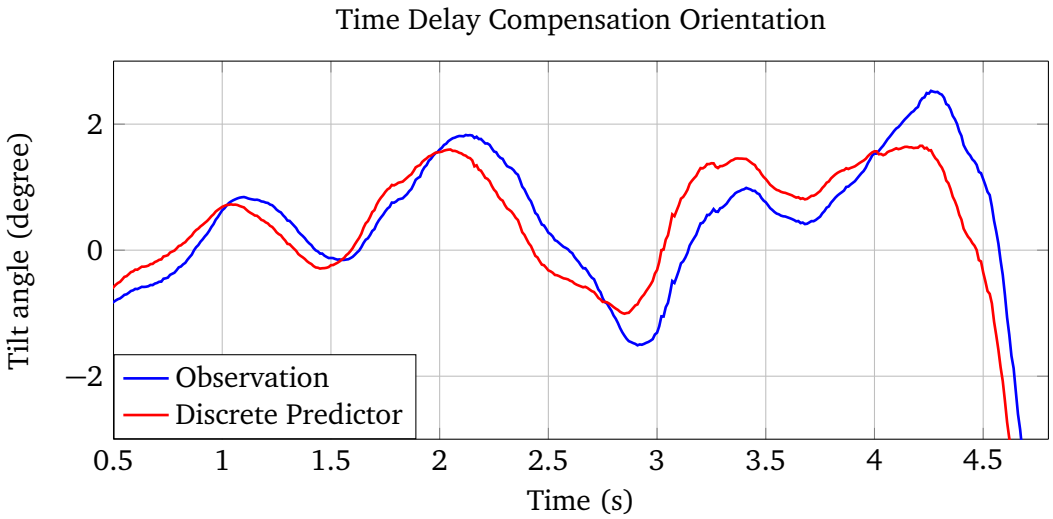


Figure 8.2: The figure presents results of tilt angle prediction. The figure consists of the discrete prediction $\mathbf{x}_{k+h}$ plotted as red line and the observations $\mathbf{z}_k$ plotted as blue line. Time delay is $t_d = 0.120s$.

that the DC gain of the plant is too small. However, in the presence of higher dynamics the DC gain of the plant changes. Thus, the prediction model is a tradeoff over the range of use. This model is not accurate enough by virtue of coupling and nonlinear dynamics.
The time delay compensation achieves reasonable results for delays smaller than $t_d < .150s$. Table 8.1 summarizes the results of the prediction scheme and shows the accuracy of the prediction (acc: $1-\frac{RMSE_{pred}}{RMSE_{del}}$). The table compares the estimated state vector $\mathbf{x}_k$ that is delayed in time with the predicted future state vector $\mathbf{x}_{k+h}$. Figure 8.3 shows the prediction results for an increased time delay of $t_d = 0.2s$.

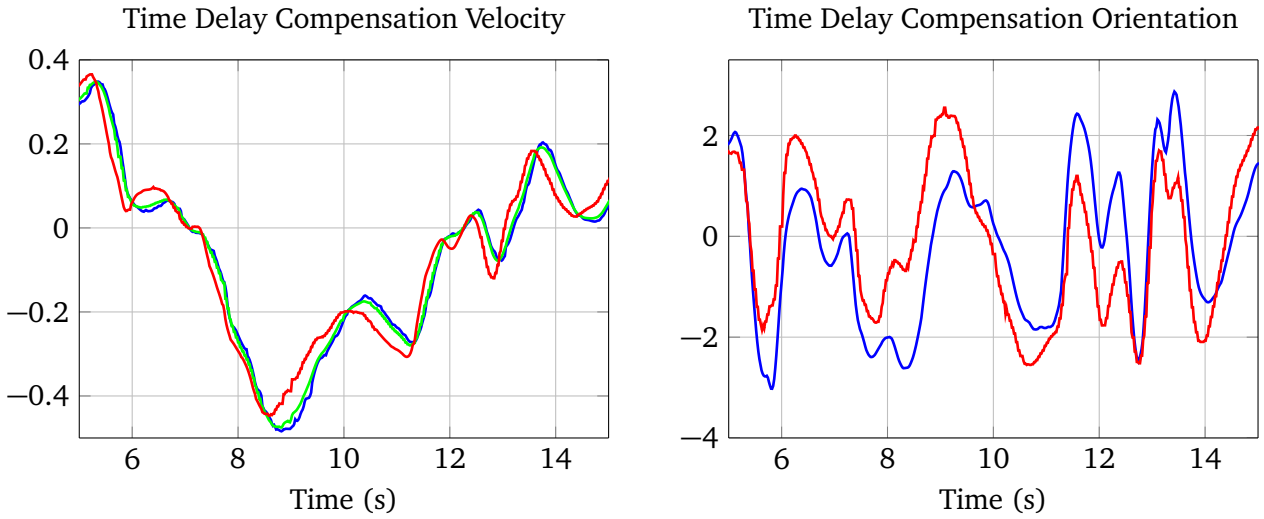


Figure 8.3: The left figure shows the velocity (left figure) and the right figure shows the orientation. Both figures represent the case of a high delay of $t_d = 0.2s$. The blue line is the observation $\mathbf{z}_k$. The green line is the estimated state vector $\mathbf{x}_k$ and the red line is the future state vector $\mathbf{x}_{k+h}$.

The future state velocity prediction produces an usable estimate of the non-time delayed state, but the deviations like overshoot start to grow. The present of an additional time delay of $t_d = 0.08s$ leads to a significantly less precisely prediction (compare $t = 12.5s$). In figure 8.3 a direct correlation between the overshoot of the velocity prediction and the poor prediction the attitude dynamic is present. For example at $t = 11s$ the tilt angle has huge overshoots. These overshoots are passed to the future state velocity prediction. At Figure 8.3 the progress of the attitude prediction fits barely with the original ob-

	$RMSE_{pred}$	$RMSE_{del}$	acc
$\dot{x}$	$0.0084m/s$	$0.0345m/s$	76%
$\dot{y}$	$0.0186m/s$	$0.0401m/s$	59%
ϕ	$0.91°$	$1.51°$	36%
θ	$1.22°$	$2.25°$	47%

Table 8.1: Quantitative results of the discrete predictor.

servation. Although the orientation prediction has a poor performance the velocity prediction of $\mathbf{x}_{k+h}$ is still reasonable. This points out that the velocity estimate mainly depends on the last velocity. The time delay compensation limited to a prediction time of $t_d = 0.20s$ due to the poor prediction performance in the case of high time delays. Beyond that, we analyze the quality of the predicted velocity estimate.

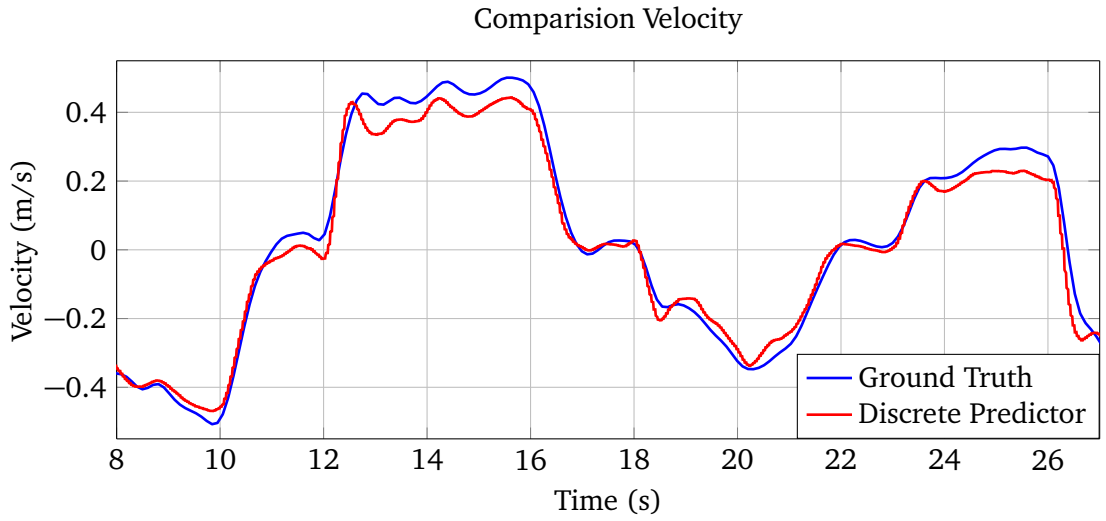


Figure 8.4: This figure compares the future state velocity estimation $\mathbf{x}_{k+h}$ with the ground truth velocity estimate from the ceiling camera system.

To do this the predicted velocities of the state vector $\mathbf{x}_{k+h}$ are compared with the ground truth velocities. The ground truth velocities are estimated by using an external camera system (the setup is explained in 7.3). Figure 8.4 indicates a good performance for the discrete predictor. The predicted velocity matches the progression of the ground truth velocity. However, some small deviations are present. First, the gain of the predicted output is smaller than the ground truth velocity. Between $t = 12 - 16s$ the predictor has a relative error of 5.3%. The discrete predictor estimates the transient behavior of the system very precisely. Deviations are only present at peaks like at $t = 10s$.

8.2 Velocity Controller

The velocity controller uses the predicated future state vector $\mathbf{x}_{k+h}$. As mentioned in the previous section the prediction of the state vector is only accurate for the translational velocities. For this reason an output feedback based on the translational velocities is implemented. Since the prediction of the attitude dynamics has a poor performance, a state feedback control scheme is inappropriate. This paragraph describes the different reference signals which were used to evaluate the velocity controller. The closed loop is evaluated with step inputs in the range of $v = 0.1 - 0.4m/s$. The step inputs were carried out in both directions. More complex maneuvers were evaluated, too. The main reference input consists of a series of step inputs. The progression is shown in figure 8.5. The du-

ration of the step input varies between $t = 2 - 4s$. Furthermore, a diagonal input is evaluated. The diagonal input consists of two reference velocities in x- and y-direction. Both have the same value.

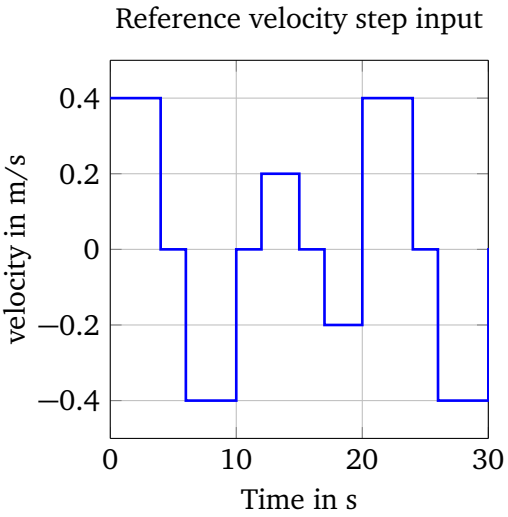


Figure 8.5: Reference progression step input

Before the new framework is evaluated, we first analyze the performance of an old PID controller without time delay compensation. The time delay of the system is clearly visible (compare figure 8.6). The closed loop reacts to a change in the reference velocity lagged in time. Two aspects catch the attention: firstly, velocity oscillations are present around the reference values (see $t = 18 - 20s$), secondly, the quadcopter is prone against disturbances ($t = 13 - 14s$). The oscillations lead to a poor performance of the closed loop. However, they are not visible to the bare eye. For instance the first five seconds of the velocity progression show that the closed loop is not exactly stationary. The system acts every sensitive against small disturbances, due to an aggressive progression of the attitude dynamics. The tilt angles fluctuate strongly to follow the reference velocity. Thus, the aim of our approach is to reduce the oscillations and to ensure an overall smoother progression of the velocity by reducing the dynamics of the close loop.

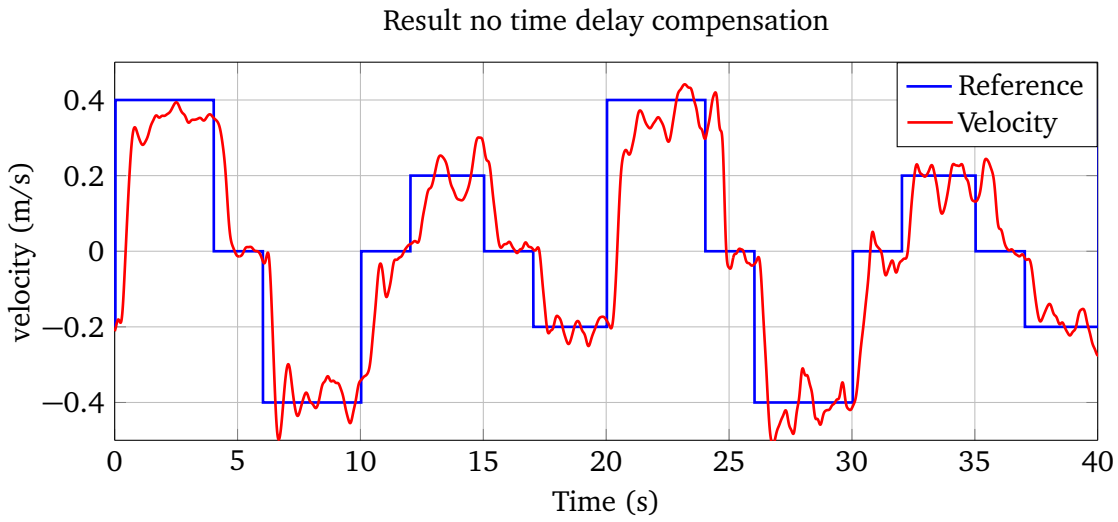


Figure 8.6: The plot shows the closed loop behavior of the velocities with an old PID controller without time delay compensation. The progression of the translational velocity in x-direction is represented by the red line and the reference velocity is plotted as blue line.

In comparison, figure 8.7 illustrates the progression of the translational velocity in x-direction based on the OP-A framework introduced in chapter 6. The average time delay in the progression is $t_d = 140ms$. The direct comparison between the old PID controller and the new OP-A based PID controller scheme indicates that the new developed algorithm has reduced the oscillations of the closed loop. The overall characteristics of the translational velocities are smoother and the closed loop seems to have a reduced dynamic. Just to conclude, the time delay compensation has reduced the small oscillations around the reference velocity. Thus, the time delay compensation works correctly and increases the performance of the closed loop. However, closed loop is still prone to disturbances but the influence of the disturbances is reduced as well as the deviations around the reference velocity ($t = 4 - 5s$). The closed loop is still

not exactly stationary (compare $t = 0 - 1s$).
The PID control scheme is active if the reference velocity is unequal to zero. In case that the reference velocity is zero the onboard hover control algorithm is activated. The controller is designed to transfer the quadcopter from a flying state into a hover state by using an off-line calculated trajectory based on dynamic inversion. The onboard controller is executed at a rate of $200Hz$ and no additional delays arise due to the fact it is executed onboard of the quadcopter. However, even the onboard controller is not capable to transfer the system without overshoot to a new reference set point ($t = 11 - 12s$ and $t = 21 - 23s$). This illustrates the limitations of the quadcopter. Figure 8.8 analyzes the change of

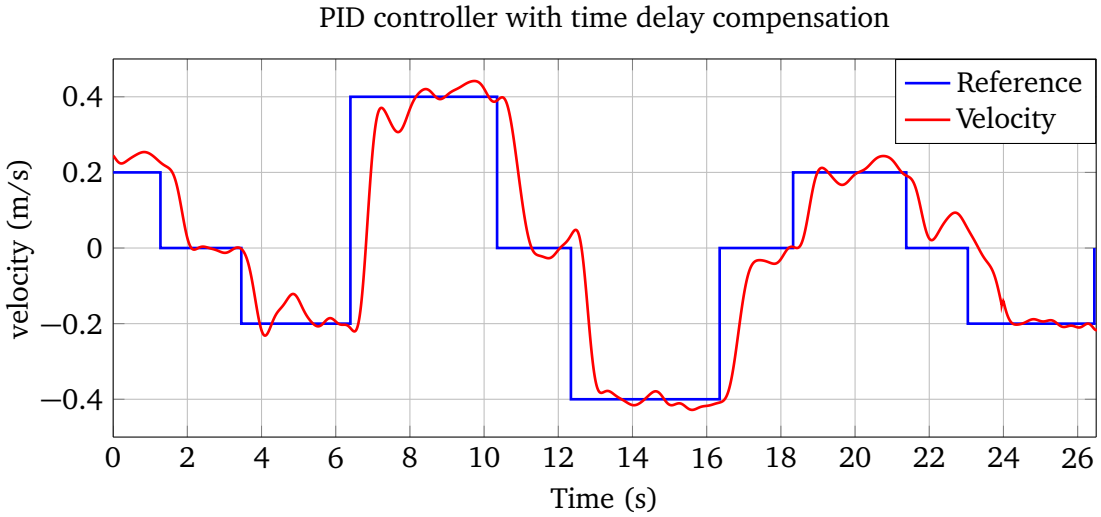


Figure 8.7: The figures shows the progression of the translational velocity in x-direction. The closed response to a series of step inputs is displayed. The figure shows the case with the new developed PID control scheme with time delay compensation.

the reference velocity from $v_{ref} = -0.2m/s$ to $v_{ref} = 0.4m/s$. To perform a change of direction the quadcopter has first to decelerate until it stops and then accelerate to the new reference velocity. During this maneuver instability can occur due to a change of direction. Therefore, this maneuver is analyzed in detail. Since it takes long to stop the quadcopter, the progression indicates that the change of direction is performed slowly. The overall duration of the maneuver is $t = 1.2s$ and therefore four times longer than a traditional change of the reference velocity. This maneuver adds more dynamic to the closed loop in such a way that the overshoot dramatically increased. Moreover, the closed loop was stable for all tested velocities from $v = -0.4m/s$ to $v = 0.4m/s$.
Figure 8.9 compares the derivative terms of the control command. In figure (a) the error signal is used and in figure (b) the derivative term is based on the velocity. The error signal derivative leads to an infinite jump at the instant a new step input is received. The control term is limited with clamping. In comparison the control command with a velocity based derivative term has a smoother progression. In both figures the tilt angle has an overshoot at the instant the step input is executed. In figure 8.9 (b) shortly after the step input was performed the tilt angle becomes negative although the control command is positive (compare $t = 1 - 2s$). Moreover, at $t = 4s$ the tilt angle suddenly drops although the control command is constant. All cases illustrate out how sensitive the quadcopter reacts to disturbances. In figure 8.9 (a) from time $t > 5s$ onwards the reference velocity is equal to zero and thus the onboard hover controller is active. The hover controller stops the quadcopter in the shortest period of time and thus the attitude dynamics change very fast. Finally, the diagonal flights are analyzed. A diagonal flight has a strong coupling between the motion of the x-axis and y-axis. A step response in one direction can arise up to 20% of the original reference velocity on the coupled axis. As figure 8.9 shows a step input induces a sudden change in the tilt angle with a high slope. In general, coupling effects are ignored in the OP-A because analysis proofed that they can be neglected even for diagonal flights, since the prediction

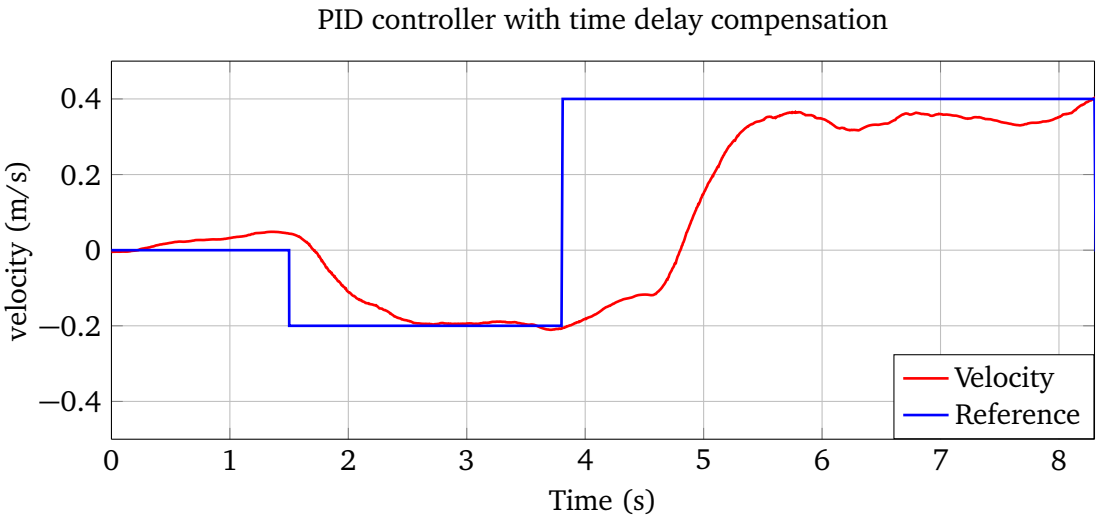


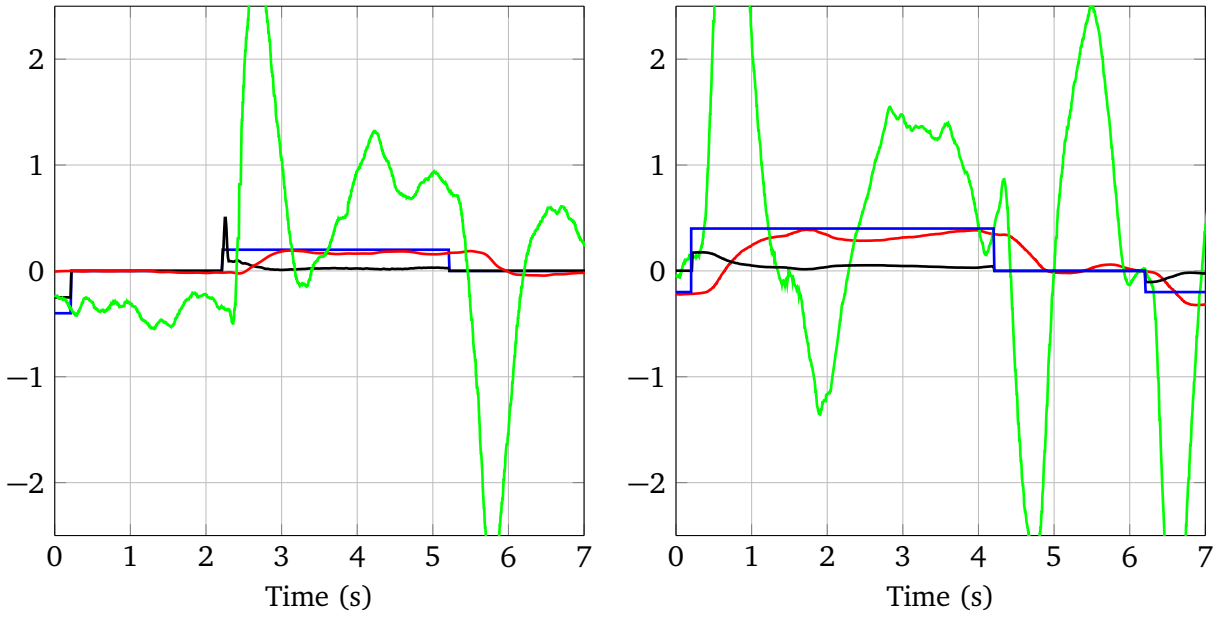
Figure 8.8: The figure shows in detail the progression how the closed loop reacts to change of direction in the reference velocity. Here, a step input from $v_{ref} = -0.2m/s$ to $v_{ref} = 0.4m/s$ is displayed.

Step input	$v_{ref} = 0.2$	$v_{ref} = 0.4$	From $v_{ref} = -0.2$ to $v_{ref} = 0.4$
Overshoot	20%	10%	22.25%
Deviation	7.5%	5.25%	5.25%
Rise time	0.43s	0.63s	1.35s

Table 8.2: Quantitative results of the velocity controller

horizon of the velocity is very short. The PID controller tries to counteract such movements but due to the structure of the PID controller as well as the mechanical structure the closed loop the motions cannot be decoupled.

This paragraph summarizes all qualitative results and table 8.2 provides a quantitative overview about the results of the velocity controller. The deviation and the overshoot of table 8.2 are presented as relative values related to the reference values. For the sake of clearance, all evaluated velocities are relative velocities based on the internal measurement of the Ar.Drone. Velocities smaller than $v = 0.10m/s$ are not controllable since the quadcopters inner control loop reacts not correctly to very small inputs. Furthermore, the variance of the IMU is 0.2°. This also indicates that small velocities and thus small tilt angles cannot be controlled correctly. In general, the results proof that the performance of the closed loop increases with higher speeds as well as the sensitivity against disturbances like wind decreases with higher velocities due to the kinematic energy of the quadcopter. However, even with the future state vector $\mathbf{x}_{k+h}$ it is not possible to control the velocity stationary precise or to counteract the disturbances. A main reason is that the Ar.Drone itself is a closed loop system and the inner control loop is unknown and cannot be modified. The velocity is controlled by actuating the roll angle and pitch angle of the quadcopter. Figure 4.6 shows that the inner control loop inaccurately controls the tilt angles and that overshoots and offsets are present. Those inaccuracies influence the performance of the velocity control loop badly.

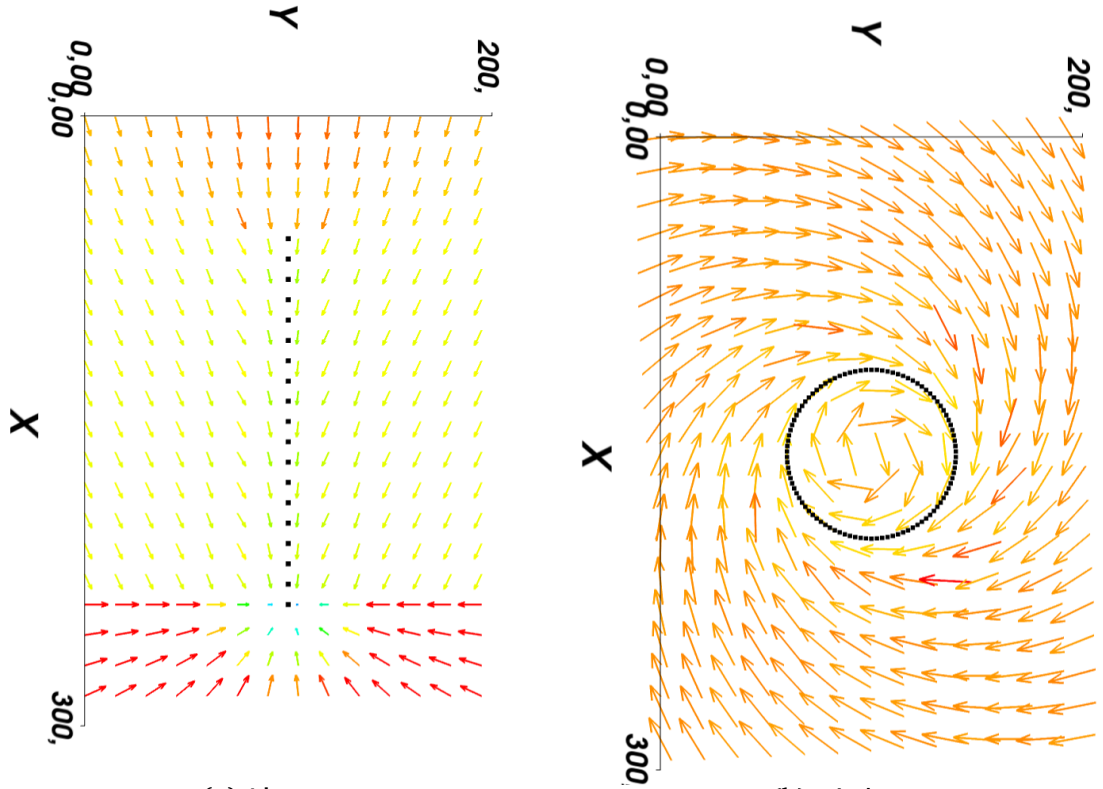


(a) Derivative Error (b) Derivative Velocity

Figure 8.9: The blue line represents the reference velocity. The red line is the measured velocity and the black line is the control command. The actuating variable of the velocity is the tilt angle (green line). Figure (a) shows an infinite jump at the time the control command takes place, only limited by the clamping of the controller. Figure (b) shows a smaller jump based on the proportional term. The tilt angles in both figures strongly fluctuate during the flight.

8.3 Path following

To evaluate the task of paths following two different kind of paths were designed. Both path are presented in the map coordinate frame. The map has a size of $300x200$ squares and the size of each square is equivalent to $1cm^2$. The origin of the frame is the top left corner.
First, the framework was tested by following a straight line. The line has a length of 190 squares that is mapped to the starting point $x = 60$ and to the final point $x = 250$. The line is straight and located in the center of the map with $y = 100$. Figure 8.10 shows the velocity field as reference vectors $\mathbf{v}_{ref}$ for each element of the map. The way-point reference path is plotted as bold black line. The velocity field is separated into three regions. The first region is the approaching area. This area is defined through a minimal distance of 30 squares from the path. The second region is the following area and is called "on track". This area features an increased α. The last region is the hovering area around the final way-point $x = 250$. All elements with $x > 250$ are part of the hover region. In the hover region $\mathbf{v}_{tan}$ is equal to zero and all velocity vectors point towards the final point $x = 250$ and $y = 100$.
The second path is a circular contour. The origin of the circle is aligned with the origin of the map at $x = 150$ and $y = 100$. The circle has a diameter of 80 squares. The velocity field of the circle rotates clockwise. The quadcopter performs in total 10 rounds for each evaluation. The circle is divided into two regions, namely, the approaching region and the on track region. The circle is small due to the limited field of view of the external camera system. The tangential velocity was scaled to $\mathbf{v}_{tan} = 0.2m/s$.
Both paths were followed with and without using the yaw-controller. The yaw-controller is used to align the center of the quadcopter with the path. Furthermore, the reference velocity of the path changes at every point. Thus, the reset modification of the integral term is disabled. Otherwise the I-term would be set to zero every time the reference velocity changes.



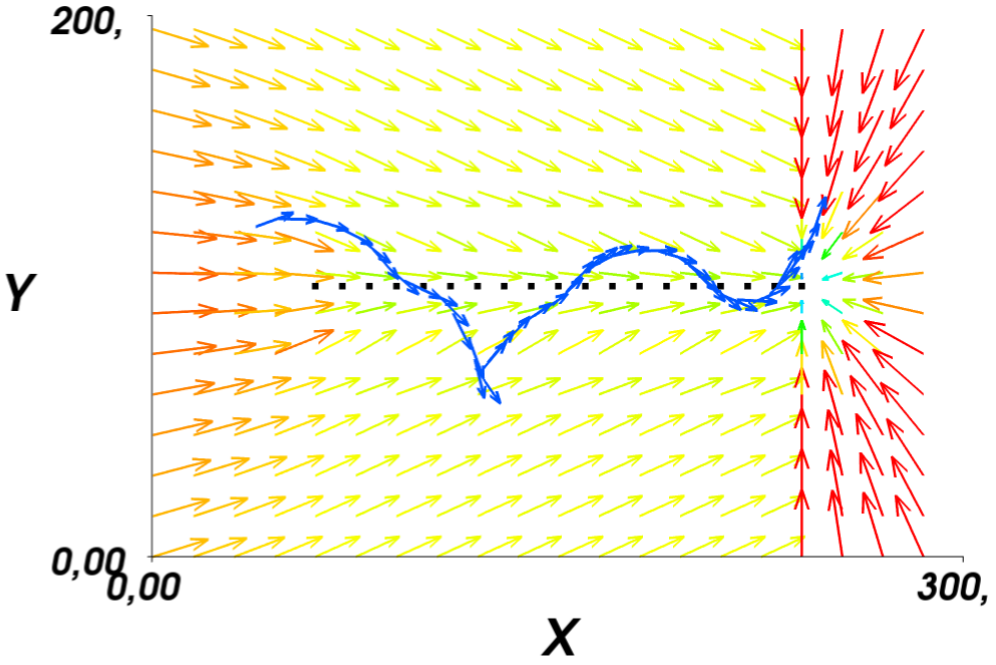
(a) Line (b) Circle

Figure 8.10: Both figures represent a reference velocity field to follow a desired path. The velocity field is a vector field and each point of the map has a vector. The black line shows the way-points of the desired path. Figure (a) shows the task of following a straight line and figure (b) shows the task of following a circle in clockwise direction.

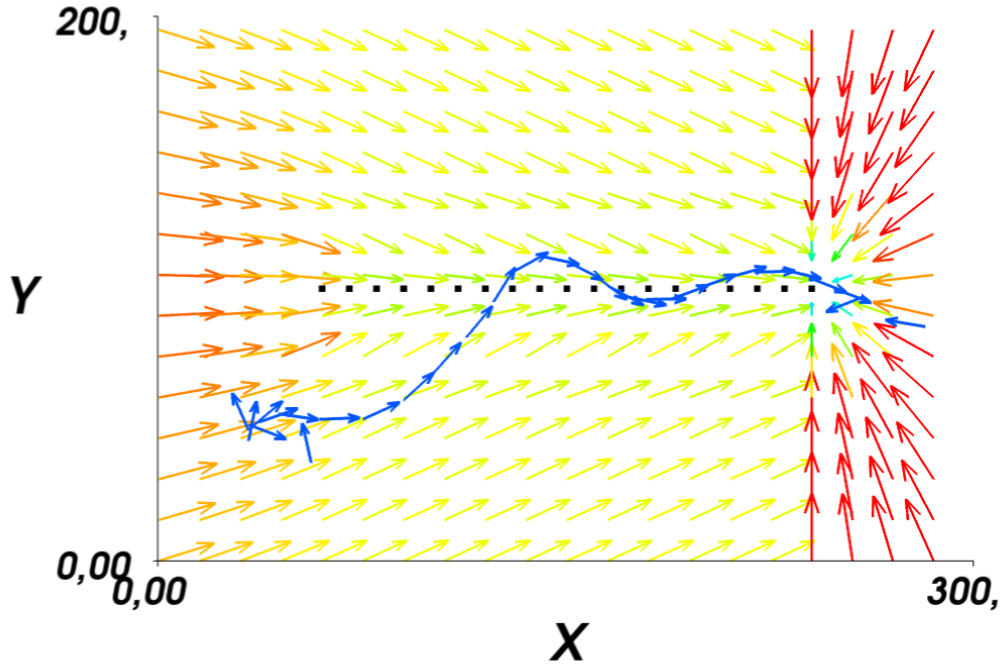
8.3.1 Path: Line

Both figures in 8.11 show how the quadcopter follows a desired path. The evaluation of the experimental flight data showed that all flights have in common that the quadcopter glides through the air because no direct friction compared to the friction of ground robots is present. The air friction is negligibly and thus the limiting factor is blade flapping. However, blade flapping is comparably small, too. The quadcopter continuously flies in one direction even if the actuating variable $\boldsymbol{u}$ is equal to zero. Moreover, very small system inputs $\boldsymbol{u} < 0.1$ are not correctly actuated and inputs $\boldsymbol{u} < 0.02$ are ignored by the quadcopter.
For instance path 1 of figure 8.11 presents the behavior if the quadcopter is controlled by using visual servoing from the external camera. In this configuration the OP-A algorithm is not used and the quadcopter is controlled at a rate of $10hz$. Due to the slow executing rate the quadcopter oscillates in y-direction with an undamped oscillation. Therefore, the use of the external camera system alone is not suitable.
The final behavior of the system is shown in path 2 of figure 8.11. The quadcopter starts at point $x = 60$ and $y = 36$. From that point onwards it approaches the path smoothly. Once the reference path is reached the path of the Ar.Drone is equivalent to a damped sin wave oscillation around the reference path. During the approach the velocity $\mathbf{v}_{aprox}$ is decreasing and hence the velocity in y-direction decreases, too. The velocity $\mathbf{v}_{tan}$ constantly points x-direction. The oscillation is induced due to the fact that the velocity controller reacts slowly to the new reference velocity and thus an overshoot occurs. The deceleration is increased due to a change of the reference velocity from the instant an overshoot is present. This happens when the quadcopter crosses the border of $y = 100$. By following the damped sin wave the error continuously decreases finally the total velocity is almost tangential and equal to $\mathbf{v}_{tan}$.

The error occurs due to the fact that the quadcopter glides through the air and thus decelerates slowly, as seen in figure 8.8 where the direction of flying was instantly changed. This illustrates a main difference between ground robot and quadcopter. The quadcopter is not exposed to rolling friction in comparison to ground robots and air drag is negligible at low speeds. At the last way-point $x = 250$ the forward flight is transformed into hovering. During this maneuver the onboard hover controller is not active. The Ar.Drone is controlled by the velocity field. The quadcopter has an overshoot in the x-direction that decelerates the movement and than flies backwards to the final point.
When summarizing all evaluated test flights, it can be seen that the average distance between the Ar.Drone and the reference path is less than $0.15m$. This holds true from the instant the reference path was approached for the first time and afterwards. During all evaluated flights, disturbances were present that pushed the quadcopter away from the track. The impact of the disturbances fluctuates between all test flights. The velocity field approach pushes the quadcopter smoothly back on track if deviations occur, instead of performing a abrupt changes of the reference velocity.
The yaw controller was evaluated during the task of following a straight line. The procedure to first align the quadcopter towards the path and afterwards to approach the path worked correctly. It is worth to note that a rotation around the z-axis adds a momentum to the quadcopter and hence the Ar.Drone starts to move in one direction during the rotation.



(a) Path 1



(b) Path 2

Figure 8.11: The two figures represent the task to follow a straight line by using a velocity field. The velocity field is plotted as velocity vector for each point of the map. The black line represent the desired path by using way-points. The blue line shows the path of the quadcopter trying to follow the desired path. Path 1 represents the case in which the Ar.Drone is completely controlled by the external camera system. The second figure represents the final results of the path following algorithm.

8.3.2 Path: Circle

All results have the following in common. Firstly, the actual yaw angle is lagged behind ψ_{ref}. The lag is so high that it can be seen by bare eye because the x-axis of the quadcopter is not aligned with the direction of the path. Secondly, the Ar.Drone measures the velocities incorrectly.

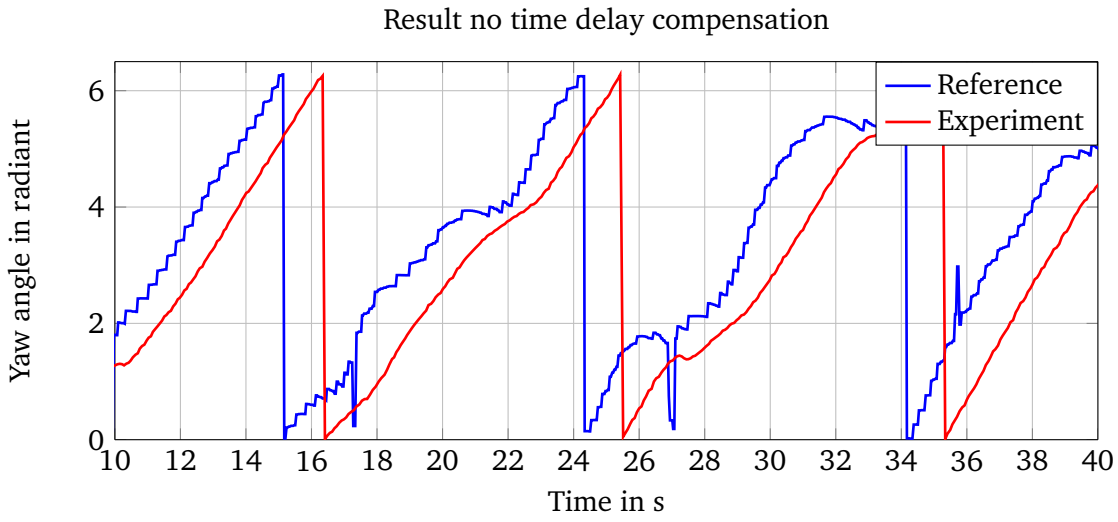


Figure 8.12: The figure shows the yaw angle measured by an external camera system (red line) and the reference yaw angle calculated by the path following algorithm.

Now, we analyze the yaw angle problem in detail. Figure 8.12 presents the time lag between the actual yaw angle and the reference yaw angle. The figure shows that the lag is huge and it verifies the assumptions made on the first sight. The plot indicates that the yaw angle adapts continuously to the new reference value. The Parrot Ar.Drone 2.0 is not capable to handle a continuously rotation as well as flying in several directions. Due to the lag in the yaw angle the quadcopter has to use both velocities to follow the path, instead of using the translational velocity in x-direction alone. A main error in the

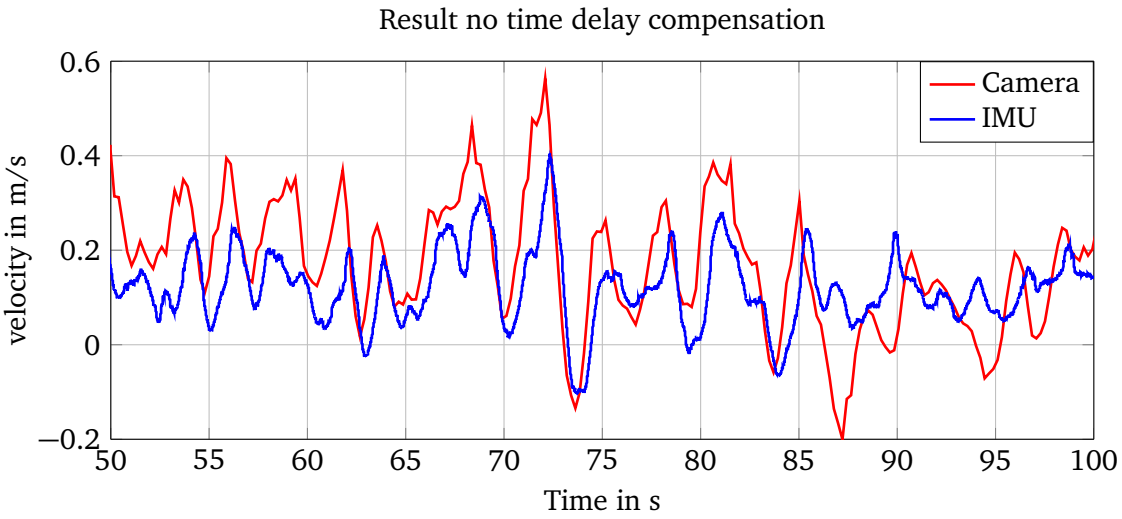


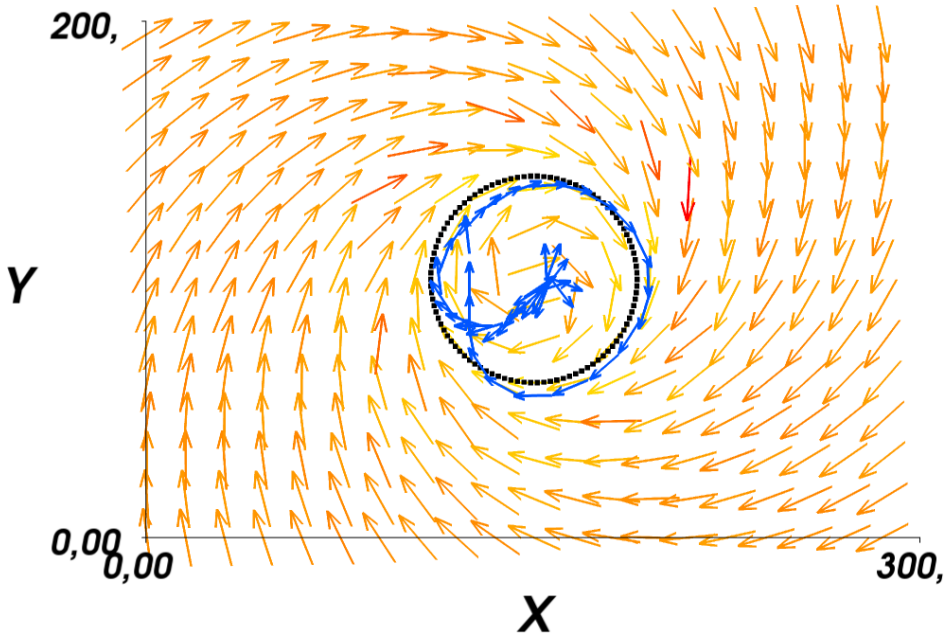
Figure 8.13: The figure shows the translational velocity in x-direction during the maneuver of flying a circle. The velocity measured onboard the Ar.Drone is drawn in blue. In addition, a ground truth velocity estimate is used. The ground truth velocity is estimated by using an external camera and drawn in red.

task to follow the circular velocity is the velocity itself. Figure 8.13 compares the velocity measured by

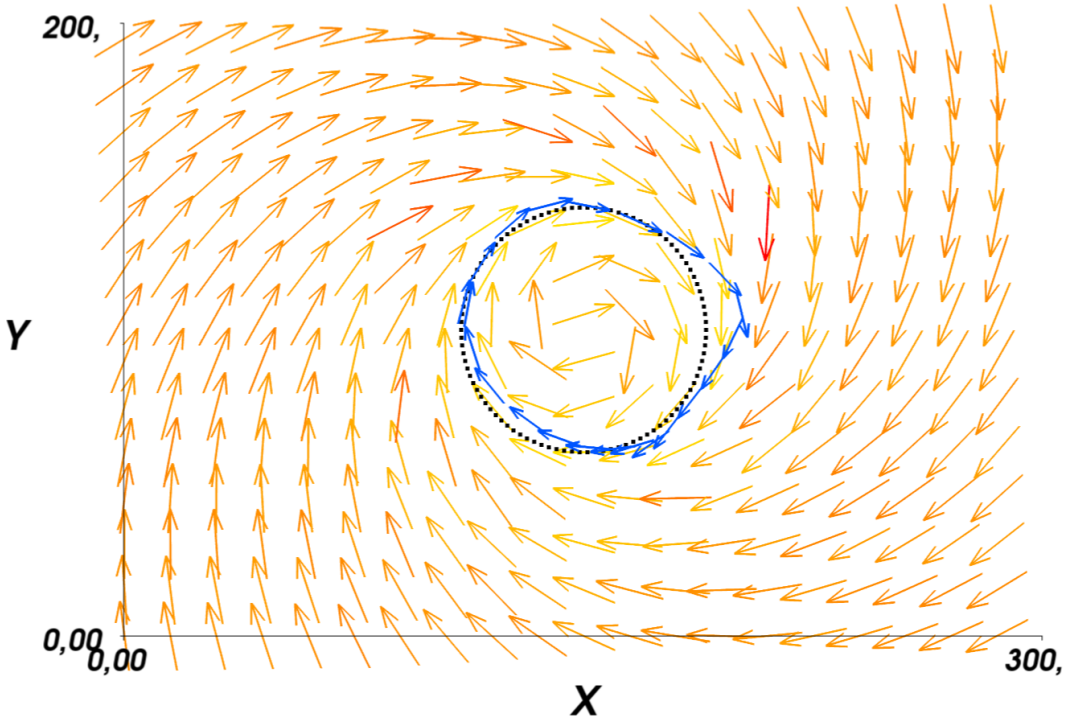
the Ar.Drone with the ground truth velocity estimated by using an external camera system. The figure illustrates that the internal velocity estimate is very inaccurate and is not able to compute the velocity correctly. The deviation between the onboard velocity and the ground truth velocity is around 36%. Moreover, the figures also indicates that under particular circumstances the velocity even has the wrong sign and hence points in the opposite direction. Both errors occur because the velocity estimation of the quadcopter is based on corner detection and optical flow detection. Those algorithms do not work correctly if the orientation of the quadcopter changes continuously. In conclusion, straight paths are followed preciser than circles. To counteract this problem, the ground truth velocity is fused in the OP-A algorithm. This reduces the errors and achieves a more accurate estimations of the velocity. However, the estimate from the camera system is overlain by noise and hence the measurement covariance is comparably high. As consequence the ground truth velocity cannot be trusted as perfect measurement. The impact of the ground truth velocity estimate is limited due to the fact it is only available every 10th spin.

The distance of the quadcopter to the next way-point on the path is quantitatively evaluated. An average circular round consists of 270 observations with an average distance of $0.11m$ from the track. Due to the lag in time of the yaw angle measurement and the errors that occur due to a continuous rotation the yaw angle control is not used in the task of follow a circular contour.

At the end of this section, two example measurements are analyzed in figure 8.14. In the first figure the quadcopter starts in the origin of the circle and then approaches the contour of the circle and follows it precisely. After three-fourths of the round the quadcopter experiences disturbances that push the quadcopter away from the reference. This leads an almost straight path of the quadcopter. If the error is within certain bounds the total velocity is almost based on the tangential velocity and hence the errors are not counteracted. The second figure clarifies this behavior. The path of the quadcopter is a snaky line around the circular contour. Small errors are almost neglected by the algorithm. Only if the error rises and reaches a certain level the velocity v_{aprox} pushes the quadcopter back on track.



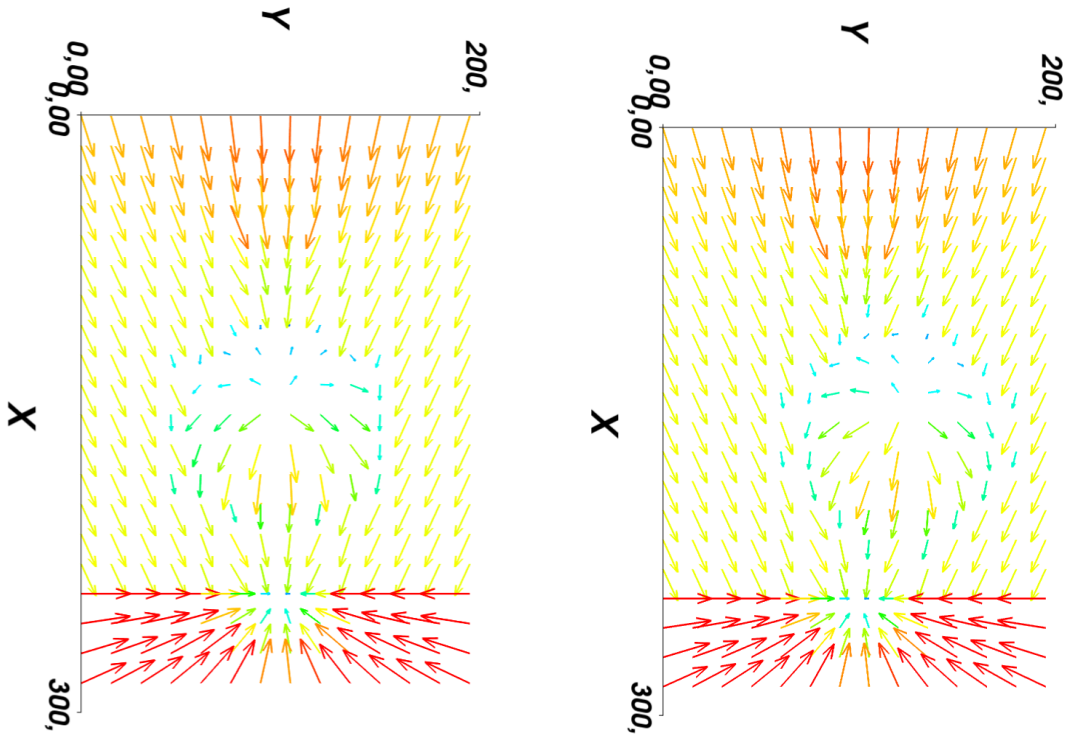
(a) Circle 1



(b) Circle 2

Figure 8.14: The two figures represent the task to follow a circular path by using a velocity field. The velocity field is plotted as a velocity vector for each point of the map. The black line represent the desired path by using way-points. The blue lines shows the way-point path of the quadcopter. Path 1 represents the case in which the quadcopter starts in the origin of the circle and then follows the contour. The second figure represents a snaky path around the desired circular path.

8.3.3 Obstacle Avoidance



(a) Obstacle center (b) Obstacle left

Figure 8.15: Both figure represent the combined velocity field consisting of path following and obstacle avoidance. Both figures show the task of following a straight line. In figure (a) the obstacle is placed in the center of the track and in figure (b) the obstacle is placed next to path.

Finally, the test case of path following is expanded by obstacle avoidance. The path is the straight line from the previous section. For the obstacle two scenarios are evaluated:

1. The obstacle is at the center of the track at $x = 150$ and $x = 100$.
2. The obstacle is next to the reference path at $x = 150$ and $y = 120$.

In both cases the object has a radius of $d = 35$ grid units. Furthermore, the quadcopter has to change the route to avoid a collision. Both cases are displayed in figure 8.15. The first scenario is more difficult to follow because the quadcopter flies directly towards the obstacle. Hence, it has to decelerate completely. The results from the last sections showed that the quadcopter reacts slowly to changes of the direction. To handle this behavior, the velocity field has to be expanded to ensure collision avoidance. So that the obstacle is virtually expanded. A detailed look at the velocity field reveals that a point exists at which the reference vector of path following $\mathbf{v}_{ref,path}$ and the reference vector of obstacle avoidance $\mathbf{v}_{ref,obst}$ cancel each other out $0 = \mathbf{v}_{ref,path} - \mathbf{v}_{ref,obst}$. In this point the total reference velocity is equal to zero. This does not only affect the center but also a contour around the cut surface of the path and the obstacle avoidance exists with a total reference velocity almost equal to zero. For this reason the quadcopter is detained in a local minimum. To avoid this minimum two solutions are possible. First, a dynamic potential field can be generated based on the old path of the quadcopter. This field prevents the quadcopter to get detained in a local minimum. This happens because based on the old position the quadcopter always has a force pushing it forward. The second solution is to define a lower bound for the reference velocity. So that velocities smaller than $v < 0.1m/s$ are scaled to $v = 0.1m/s$.

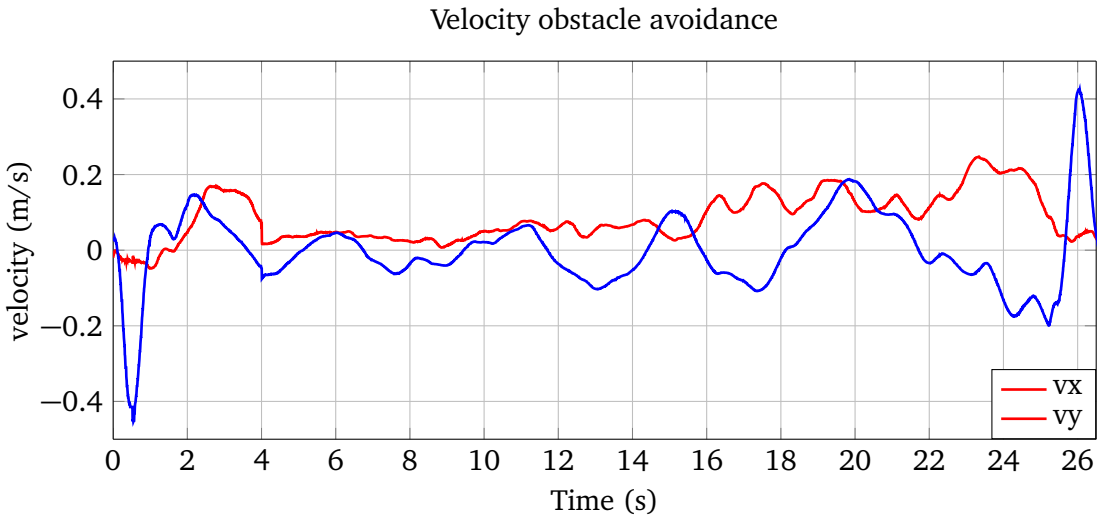
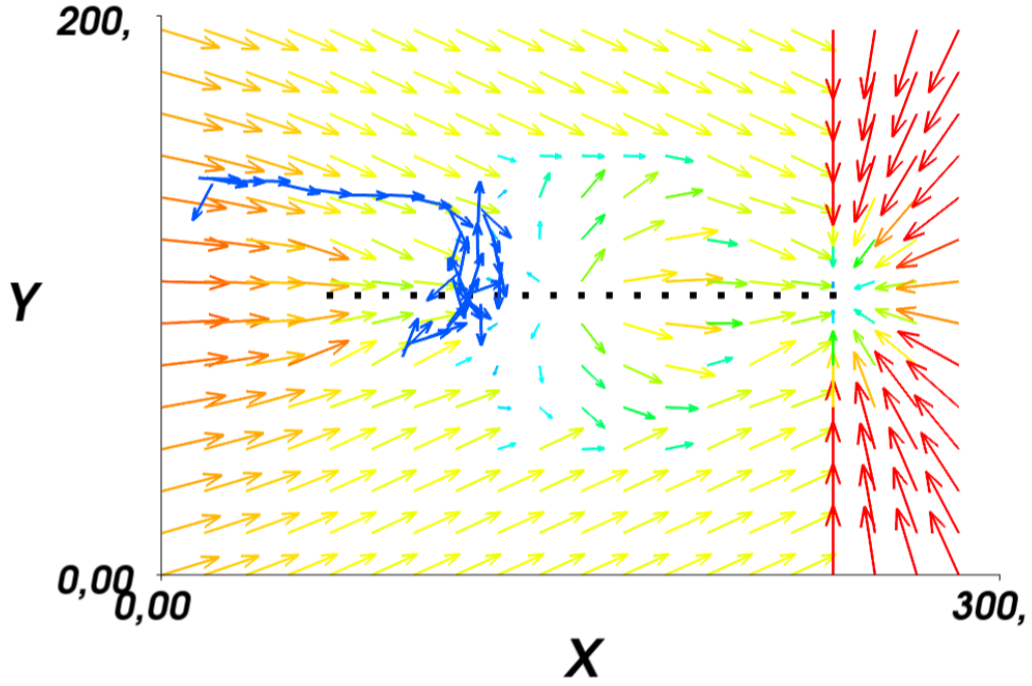
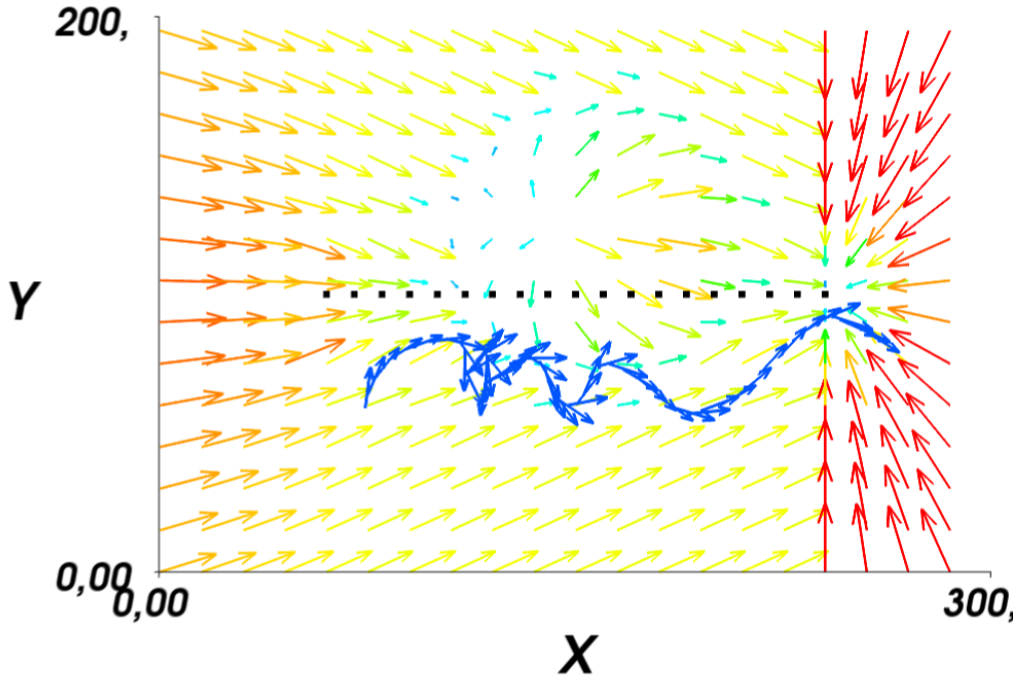


Figure 8.16: The figure represents the velocities related to the path of the quadcopter in the velocity field 8.17. The total duration of this flight was around 30s. The figure shows a long time span of $t = 6s$ in which the quadcopter has almost no tangential velocity and is fixed in a local minimum of the velocity field.

To avoid this behavior and to proof the concept of path following and obstacle avoidance for quadcopter the second case is analyzed. Figure 8.17 shows the path of the quadcopter during the task of following a straight line. The velocities related to this maneuver are presented in 8.16. The quadcopter first approaches the path in the first four seconds and around $x = 100$ or $t = 4s$ the quadcopter suddenly decelerates to almost $v_x = 0$ due the fact that the quadcopter is exposed the obstacle velocity field. For the time span of the next six seconds the quadcopter flies slowly forwards and flies simultaneously to the left and right side because in the local minimum impulses from all directions are present (compare to the region where the path vectors point in all direction $x = 100$ and $y = 90$). By doing so, the quadcopter leaves the local minimum and flies slowly in tangential to the path $v_{tan} = 0.1m/s$. The quadcopter oscillates in y-direction alongside the contour between the obstacle velocity field and the path following velocity field. The position and velocity oscillate. The oscillations have a maximum amplitude of $\hat{y} = 0.1m$. At point $x = 150$ the reference vector of the obstacle avoidance changes directions and accelerates the quadcopter. Simultaneously the quadcopter tries to approach the path again.
If obstacle is not located in the center of the track the algorithm successfully avoids a collision with the obstacle and does not get stuck in a local minimum. However, during all test flights oscillations of the quadcopter on the circular contour were present. This is a well known behavior of potential field based navigation.



(a) Obstacle center



(b) Obstacle left

Figure 8.17: The two figures represent the task to follow a path and to perform collision avoidance simultaneously. The velocity field is plotted as velocity vector for each point of the map. The black line represent the desired path by using way-points. The blue line shows the path of the quadcopter. Path 1 represents the case in which the obstacle is located in the center of the map. Thus, the velocities of path following and obstacle avoidance cancel each other out. The Ar.Drone got stuck in a local minimum. The second figure presents the case to avoid a obstacle next to the path. Here, the quadcopter arises out of the local minimum and. flies slowly along the contour of the circle.

9 Conclusion

In this thesis a velocity based control concept for the Parrot Ar.Drone 2.0 was developed to contribute to the field of autonomous flying in indoor environments. The proposed framework is modular and allows other researchers to develop their own applications for a velocity control.
To achieve this goal, velocity controller in conjunction with an observer-prediction algorithm (OP-A) was developed. The OP-A receives measurements from the Parrot Ar.Drone 2.0. these observations have the following features: the measurements arrive in bunches of up to 8 messages, they may have the same time stamp, they have tiny gaps up to $40ms$ between each other. Therefore, the observer of the OP-A fuses the different measurements and synchronizes the observations in time.
A time delay analysis of the closed loop identified that the low-cost IMU and onboard velocity measurement unit have huge delays that are not negligible. In addition, the wireless data transport adds non-deterministic delays as well as packet losses. To compensate the inherent time delays up to $t_d = 250ms$, a discrete h-step ahead predictor was developed. The discrete predictor as well as the observer use a linear model of the quadcopter dynamics. The results indicate that the attitude dynamics are predicted poorly because of to the strong nonlinear couplings between both axis. A master-thesis from TU Delft [20] performed a Grey-box identification of the Parrot Ar.Drone 2.0. They assumed a linear model as well and achieved similar results. This points out the limitations of the linear model. Further investigations should focus on the identification of the nonlinear attitude dynamics. The prediction of the translational velocities is precise. Moreover, experiments indicated no beneficial improvements when a nonlinear dynamic model for the translational velocities was used.
The velocity controller in conjunction with the OP-A successfully eliminated the oscillations around the reference values that were induced by the time delay of the closed loop. The novel approach increased the overall performance of the closed loop, added stability to the system, and facilitates the velocity control of the quadcopter from a ground station computer. Even instant changes of the direction are possible and do not lead to instability. However, the quadcopter is still prone to disturbances and hence deviations around the reference velocities occur. They are caused due to changes of tilt angles, induced by the onboard controller of the quadcopter. Hence, it is not possible to counteract such deviations. The quadcopter is assumed as black box and no information related to the inner control loop is available. The lack of information indicates the limitations of the closed loop system.
Finally, in this thesis we applied the theory of velocity field control successfully to the field of quadcopters. Therefore, a way-point based path was transformed into a velocity field. The velocity field approach enables the quadcopter to follow a reference path smoothly. Owning to the velocity field that points always in the direction of the path, disturbances are counteracted more effectively. In addition, a yaw angle controller is implemented. This feature allows the quadcopter to align the x-axis of the quadcopter with the path while performing the task of path following.
We expanded the task of path following to perform obstacle avoidance. A obstacle is defined as a Gaussian repulsion field which was converted into a velocity field. The path following velocity field is overlain by the velocity field created by the obstacle. The challenges that occurred in the obstacle avoidance can be summarized into two groups. Firstly, the quadcopter glides through the air and thus decelerates slowly if the actuating variable is zero. Secondly, the overlaid velocity field has regions in which the velocity is equal to zero because both fields cancel each other out. The quadcopter can get detained in the local minimum. We mastered those challenges and applied the obstacle avoidance successfully to quadcopters.
To perform the task of path following as well as obstacle avoidance, the position of the quadcopter is required. Therefore, an external camera system was used to estimate the pose of the quadcopter. The

additional pose estimation was used to calculate a ground truth velocity. This observation is merged with the observations from the quadcopter in the OP-A algorithm to achieve more accurate results.
Just to conclude, during the work of thesis we not only provided a velocity controller with an integrated framework to compensate the inherent time delays, and to synchronize the observations, but we also successfully applied the concepts of path following and obstacle avoidance based on velocity field to quadcopters. We see a great potential in this concept and achieved good results in conjunction with our developed velocity controller.

10 Future work

For the presented framework a number of different research possibilities are of interest. The possible extensions and modifications are summarized as follows:

1. The **Quadcopter model** is reasonable accurately for the translational dynamics. But the attitude dynamics are only modeled badly by the linear model. Therefore, it would be of interest to identify a nonlinear attitude model that explicitly models the coupling between the x-axis and y-axis.

2. The **time delay** is dynamically computed based on the actual echo ping. A better result can be gained if the ping is predicted a short time span ahead to detect a bad connection before it will occur. This allows the framework to transfer the quadcopter into hover mode before the connection will get lost. Researchers in [35] analyzed different methods to predict the time delay of data transportation in a wireless network.

3. It would be of great interest to evaluate different **controllers**. The use of a state feedback controller is difficult due to the poor prediction quality of the attitude dynamics. However, new control concepts in H_{inf} as well as model predictive control (MPC) are interesting strategies to investigate. A promising approach is dynamic inversion. Based on the transfer function between the tilt angle input and the velocity output dynamic inversion can be used to calculate the reference tilt angle to reach the desired velocity. Moreover, dynamic inversion is already used in the off-line path planning algorithm integrated in the Ar.Drone.

4. The onboard **velocity measurement** of the Parrot Ar.Drone 2.0 is based on optical flow and corner detection from the downward facing camera. Our experiments indicate that a straight path is estimated correctly but errors occur if the quadcopter is continuously rotating around the z-axis (circular contour). Therefore, a possible research topic could be to use the developed velocity controller in conjunction with PTAM to estimate the ground truth pose and velocity of the quadcopter.

5. Finally, the task of **velocity field control** has a great potential for further research. The results of this work illustrated that the quadcopter can be detained in a local minimum. This behavior is well known from ground robots. This points out, that the same problems which are present in ground robot emerge es well in quadcopter velocity field navigation. Several methods can be applied to tackle the problem of a local minimum. For instance, the reference vector of the velocity field should be scaled to a minimum velocity of $\mathbf{v}_{min} = 0.1m/s$ because slower velocities are no probably controlled. After that, a dynamic potential field can be integrated based on the path the quadcopter has taken. This potential field leads to a reference velocity that always points forward and thus arises the quadcopter out of the local minimum. Moreover, it is of interest to achieve a smooth path of the quadcopter, instead of oscillating between the reference vectors of $\mathbf{v}_{ref,path}$ and $\mathbf{v}_{ref,obst}$. Therefore, the conjunctions of both fields have to be modified. Last but not least, it would be of interest to adapt the algorithm to different types of quadcopters.

Bibliography

[1] Parrot. Developer Guide SDK 1.7. 2012.

[2] Pierre-Jean Bristeau, François Callou, David Vissière, and Nicolas Petit. The Navigation and Control technology inside the AR . Drone micro UAV. *Proceedings of the 18th IFAC World Congress, 2011*, 18:1477–1484, 2011.

[3] Charles River Analytics. Package:Robot Localization, 2014. http://wiki.ros.org/robot_localization

[4] Kenzo Nonami, Farid Kendoul, Wei Wang, and Daisuke Nakazawa Satoshi Suzuki. *Autonomous Flying Robots*. Springer, 2010.

[5] Robert Seifried and Optimal Design. *Dynamics of Underactuated Multibody Systems*.

[6] Gm Hoffmann, H Huang, and Sl Waslander. Quadrotor helicopter flight dynamics and control: Theory and experiment. *American Institute of Aeronautics and Astronautics*, pages 1–20, 2007.

[7] Control Design Ideas. Introduction to Feedback Control of Underactuated VTOL Vehicles. (february), 2013.

[8] Pierre-jean Bristeau and Nicolas Petit. Presentation Parrot Ar.Drone2, Inside the Ar.Drone Micro UAV. Technical report, 2011.

[9] A.G Fallis. AR-Drone as a Platform for Robotic Research and Education. *Journal of Chemical Information and Modeling*, 53(9):1689–1699, 2013.

[10] Tang. *Vision-Based-Control for autonomous quatrotor UAVs*. PhD thesis, Princeton, 2013.

[11] Perry Y Li and Roberto Horowitz. Passive Velocity Field Control (PVFC): Part 2 Application to Contour Following. 2001.

[12] Perry Y Li and Roberto Horowitz. Passive Velocity Field Control (PVFC): Part 1 Geometry and Robustness. 2001.

[13] Advanced Intelligent and Mechatronics Proceedings. Closed Loop Control of 3D Underactuated Vehicles via Velocity Field Tracking. (July):355–360, 2001.

[14] Jakob Julian Engel. *Autonomous Camera-Based Navigation of a Quadrocopter*. PhD thesis, Technische Universität München, 2011.

[15] Jakob Engel and Daniel Cremers. Scale-Aware Navigation of a Low-Cost Quadrocopter with a Monocular Camera. *Robotics and Autonomous Systems*, 62(11):1646–1656, 2014.

[16] Anusha Gururaj Jamkhandi, Saee Tulpule, Ashvini Chaturvedi, and Jean-noel Charvet. Controlling the Position and Velocity in Space of the Quad-Rotor UAV AR . Drone Using Predictive Functional Control and Image Processing in Open CV. *International Conference on Signal Processing Systems (ICSPS 2012)*, 58(Icsps):14–18, 2012.

[17] Parrot. Product Information Parrot Ar.Drone 2.0, 2016.

[18] ROS Wiki, 2016. http://wiki.ros.org/

[19] ROS-REP 103, 2016. http://www.ros.org/reps/rep-0103.html

[20] Qianying Li. Grey-Box System Identification of a Quadrotor Unmanned Aerial Vehicle. 2014.

[21] H. Zhang L. Xie. *Control and Estimation of Systems with Input/Output Delays*. Springer.

[22] E.F. Camacho J.E. Normey-Rico. *Control of Dead-time Processes*. Springer, 2007.

[23] P. Bouffard. On-board Model Predictive Control of a Quadrotor Helicopter: Design, Implementation, and Experiments. page 67, 2012.

[24] Franz Hover. 9 Kinematics of Moving Frames. *Course Handout_MIT*, 2009.

[25] Philippe Martin and Erwan Salaün. The true role of accelerometer feedback in quadrotor control. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1623–1629, 2010.

[26] Dinuka Abeywardena, Sarath Kodagoda, Gamini Dissanayake, and Rohan Munasinghe. Improved State Estimation in Quadrotor MAVs: A Novel Drift-Free Velocity Estimator. pages 1–8.

[27] Package: ardrone_autonomy, 2014.

[28] R Maximo. Identification and PID Control for a Quadrocopter. pages 77–82, 2014.

[29] Keqin Gu and Silviu-iulian Niculescu. Survey on Recent Results in the Stability and Control of Time-Delay Systems *. 125(June 2003), 2016.

[30] Jean-pierre Richard. Time-delay systems: an overview of some recent advances and open problems. 39:1667–1694, 2003.

[31] Svante Björklund. *A Survey and Comparison of Time-Delay Estimation Methods in Linear Systems*. Number 1061. 2003.

[32] O Sename. Analysis, observation and control of time-delay systems. Technical Report September, 2014.

[33] Stephen Armah and Sun Yi. Altitude Regulation of Quadrotor Types of UAVs Considering Communication Delays. *IFAC-PapersOnLine*, 48(12):263–268, 2015.

[34] Hu Wang. *Dynamics of Controlled Mechanical Systems with Delayed Feedback*. Springer, 2002.

[35] Zhong-da Tian, Shu-jiang Li, Yan-hong Wang, and Hong-xia Yu. Networked Control System Time-Delay Compensation Based on Time-Delay Prediction and Improved Implicit GPC. pages 3–18, 2015.

[36] George Nikolakopoulos and Kostas Alexis. Switching Networked Attitude Control of an Unmanned Quadrotor. 11:389–397, 2013.

[37] Rachana A Gupta and Mo-yuen Chow. 1 Overview of Networked Control Systems. pages 1–24.

[38] Richard M Murray and Technology Draft. An Introduction to Networked Control Systems. (March), 2006.

[39] Local Interaction and Global Interaction. Multisensor D 25. pages 1–27.

[40] Lindsay Kleeman. Understanding and applying Kalman filtering. *Proceedings of the Second Workshop on Perceptive . . .* , pages 1–37, 1996.

[41] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):1999–2000, 2002.

[42] T.D. Larsen, N.a. Andersen, O. Ravn, and N.K. Poulsen. Incorporation of time delayed measurements in a discrete-time Kalman filter. *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171)*, 4(December):3972–3977, 1998.

[43] Wei Li, Tianguang Zhang, and Kolja Koehnlenz. A vision-guided autonomous quadrotor in an air-ground multi-robot system. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2980–2985, 2011.

[44] A Gonzalez, Garcia Pedro, Pedro Albertos, Pedro Castillo Garcia, and R Lozano. Robustness of a discrete-time predictor-based controller for time-varying measurement delay. 2012.

[45] Laszlo Kis and Bela Lantos. Time-delay extended state estimation and control of a quadrotor helicopter. *2012 20th Mediterranean Conference on Control & Automation (MED)*, pages 1560–1565, 2012.

[46] Ricardo Sanz, Pedro Garc, and Pedro Castillo. Time-delay Compensation Using Inertial Measurement Sensors for Quadrotor Control Systems.

[47] P Garcia. Robust prediction-based control for unstable delay systems. (December):4–9, 2003.

[48] G.C.Y. Li, Y. , Ang, K.H. , Chong. PID control system analysis and design. 26(November):32–41, 2007.

[49] Miad Moarref, Camilo Ossa-Goemez, and Luis Rodrigues. Multiloop controller design for a fly-by-wireless UAV quadrotor based on a multirate sampled-data model. *IEEE Transactions on Aerospace and Electronic Systems*, 51(3):2319–2331, 2015.

[50] Setpoint Weighting und Anti-Windup-Methoden. Technical report, TU Ilmenau, 2010.

[51] Antonio Visioli. *Practtical PID Control*. Springer, 2006.

[52] Thomas Moore and Daniel Stouch. A Generalized Extended Kalman Filter Implementation for the Robot Operating System. 2014.

[53] Saman Khodaverdian, Lukas Klodt, and Volker Willert. Motion Control for UAV-UGV Cooperation with Visibility Constraint.

[54] C. R. McInnes. Velocity field path-planning for single and multiple unmanned aerial vehicles. *Aeronautical Journal*, pages 419–426., 2003.

[55] Omer Cetin, Ibrahim Zagli, and Guray Yilmaz. Establishing Obstacle and Collision Free Communication Relay for UAVs with Artificial Potential Fields. In *Intell Robot Syst*, pages 361–372. Springer, 2013.

[56] Perry Y Li and Roberto Horowitz. Passive Velocity Field Control of Mechanical Manipulators. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 15(4):751–763, 1999.

[57] Seunghan Lim, Yeongju Kim, Dongjin Lee, and Hyochoong Bang. Standoff Target Tracking using a Vector Field for Multiple Unmanned Aircrafts. pages 347–360, 2013.

[58] Matevž Bošnak, Drago Matko, and Sašo Blažič. Quadrocopter hovering using position-estimation information from inertial sensors and a high-delay video system. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 67(1):43–60, 2012.

[59] A Sanchez. A Passive Velocity Field Control for Navigation of Quadrotors with Model-free Integral Sliding Modes. *Intell Robot Syst*, pages 373–385, 2014.

[60] Changsu Ha, Francis B Choi, and Dongjun Lee. Preliminary Results on Passive Velocity Field Control of Quadrotors. In *International Conference on Ubiquitous Robots and Ambient Intelligence*, number 1, pages 378–379, 2012.

[61] BOSCH USA. Package: usb_cam, 2016. http://wiki.ros.org/usb_cam

[62] Vincent Rabaud. Packageimage_proc, 2016. http://wiki.ros.org/image_proc

[63] Hamdi Sahloul. Package: ar_sys, 2014. http://wiki.ros.org/ar_sys