View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Lund University Publications - Student Papers

Navigation and Autonomous Control of a Hexacopter in Indoor Environments

Johan Fogelberg

LUND UNIVERSITY

Department of Automatic Control

Msc Thesis
ISRN LUTFD2/TFRT--5930--SE
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

© 2013 by Johan Fogelberg. All rights reserved.
Printed in Sweden by Media-Tryck
Lund 2013

Contents

Abstract

This thesis presents methods for estimation and autonomous control of a hexacopter which is an unmanned aerial vehicle with six rotors. The hexacopter used is a ArduCopter 3DR Hexa B and the work follows a model-based approach using Matlab Simulink, running the model on a PandaBoard ES after automatic code generation. The main challenge will be to investigate how data from an Internal Measurement Unit can be used to aid an already implemented computer vision algorithm in a GPS-denied environment.

First a physical representation is created by Newton-Euler formalism to be used as a base when developing algorithms for estimation and control. To estimate the position and velocity of the hexacopter, an unscented Kalman filter is implemented for sensor fusion. Sensor fusion is the combining of data from different sensors to receive better results than if the sensors would have been used individually. Control strategies for vertical and horizontal movement are developed using cascaded PID control. These high level controllers feed the ArduCopter with setpoints for low level control of angular orientation and throttle.

To test the algorithms in a safe way a simulation model is used where the real system is replaced by blocks containing a mix of differential equations and transfer functions from system identification. When a satisfying behavior in simulation is achieved, tests on the real system are performed.

The result of the improvements made on estimation and control is a more stable flight performance with less drift in both simulation and on the real system. The hexacopter can now hold position for over a minute with low drift. Air turbulence, sensor and computer vision imperfections as well as the absence of a hard real-time system degrades the position estimation and causes drift if movement speed is anything but very slow.

Preface

This thesis was made in 2013 by myself, Johan Fogelberg, supported by my company supervisor Simon Yngve and university supervisor Prof. Anders Robertsson. The work was carried out at Combine Control Systems AB in cooperation with Lunds Tekniska Högskola (LTH), Department of Automatic Control, both located in Lund, Sweden. This thesis is a sequel to a previous thesis work [1] at Combine Control Systems by Niklas Ohlsson and Martin Ståhl and begins where their work ended.

Acknowledgments

First I would like to thank my company supervisor Simon Yngve who have guided me through this thesis work, providing both knowledge and ideas. I also want to thank my university supervisor Prof. Anders Robertsson for his help and knowledge and Mathworks employee Daniel Andersson for support and shown interest. Finally I want to thank Amanda Hagbrand for her never ending support and trust in me.

List of symbols

A	hexacopter cross-sectional area
C	friction constant
F	forces affecting the hexacopter
P	state estimate covariance matrix
I	inertia tensor
J_r	propeller rotational inertia
Q	process noise covariance matrix
R	measurement noise covariance matrix
R_B^E	body to earth rotation matrix
R_E^B	earth to body rotation matrix
V	linear velocity
b	thrust constant
d	drag factor
g	acceleration due to gravity
l	hexacopter arm length
m	hexacopter total mass
p	roll rate, body frame
q	pitch rate body frame
r	yaw rate, body frame
u	linear velocity along x-axis, body frame
v	linear velocity along y-axis, body frame
w	linear velocity along z-axis, body frame
x	linear position along x-axis
y	linear position along y-axis
z	linear position along z-axis
Ω	propeller speed

Ω_r	overall propeller speed
θ	pitch angle
μ	rotor drag coefficient
ρ	air density
τ	torques affecting the hexacopter
ϕ	roll angle
ψ	yaw angle
ω	angular velocity

Acronyms

EKF	Extended Kalman Filter
GPS	Global Positioning System
IMU	Inertial Measurement Unit
PID	Proportional Integral Derivative
PWM	Pulse Width Modulation
SRUKF	Square Root Unscented Kalman Filter
UAV	Unmanned Aerial Vehicle
UKF	Unscented Kalman Filter
VTOL	Vertical Take-Off and Landing
wrt	with respect to

1. Introduction

1.1. Overview

A multirotor is a type of aircraft similar to the traditional helicopter but with more than two rotors. The most common amount of rotors are 3 (tricopter), 4 (quadcopter), 6 (hexacopter) or 8 (octocopter), but any configuration is possible. More rotors give a higher maximum lifting capacity but are more expensive to build and need a higher current output from the batteries. If the multirotor is carrying expensive equipment 6 or especially 8 rotors can be recommended since a crash can be avoided even if one motor fails during flight. An octocopter can experience a motor failure and still have full control while a hexacopter will loose control of its heading. Multirotors have attracted a lot of attention for research in recent years due to their maneuverability, simple construction, flexibility and ability to take a payload. Today their main commercial use is related to aerial photography, surveillance and remote sensing [2]. With a few exceptions all multirotors belong to the families of vertical take-off and landing (VTOL) and unmanned aerial vehicles (UAV) and they can either be controlled manually with a radio controller or operate autonomously on their own.

When flying autonomously outdoors the Global Positioning System (GPS) is usually the main source of information for position estimation, making navigation indoors a complex task since no GPS signal is available. To cope with this the most common approach is to use computer vision systems with cameras or laser scanners. Due to the computational complexity of position and velocity estimation using these systems, a low sampling rate and/or external processing is often needed. For research purposes a stationary motion capture system can be used to get accurate position and velocity estimates but this can of course only be used at a predetermined location.

In this thesis autonomous navigation and control of a hexacopter will be presented using only on-board sensors and on-board processing.

1.2. Previous work

This thesis begins where a previous thesis [1] by Niklas Ohlsson and Martin Ståhl ended. The main scope of [1] was to use a model based approach to implement

computer vision algorithms, position control and autonomous behavior in a GPS-denied environment. A PandaBoard, which is a software development platform, was attached to the hexacopter and was used to run a Matlab Simulink model. Below is a list of work that was carried out in their thesis that will be used in this thesis.

- Assembling of all hardware
- Creating a communication protocol for the Ethernet-connection between the PandaBoard and the hexacopter autopilot.
- Developing computer vision algorithms for position estimation.
- Developing a camera simulator for the web camera that is used.
- System identification of the relation between setpoint and measured value of the angular rotations controlled by the hexacopter autopilot.
- Choosing the Navigation-Guidance-Control architecture of the Matlab Simulink model.

1.3. Problem formulation and objectives

One of the most important tasks in [1] was to attain a stable hover and this was partly accomplished. The hexacopter could stay in position for some time but was subject to drift and was not very stable, mainly because of some of the weaknesses in the vision algorithm, unsatisfying altitude hold and Simulink lags.

The vision algorithm is based on template matching where a small piece of an image frame, called template, is chosen and then by trying to fit this template into the next image frame a pixel displacement can be calculated. But if no good match can be found, a small piece of the most recent image is chosen as a new template and this gives a small time delay. By looking at logged data it is clear that this happens very often causing moments of blackout in the position estimation and thereby increased drift in the position estimation.

In [1] the altitude was controlled by the ArduCopter autopilot (see Sec. 2.1 for more details). The performance of the altitude controller was not satisfactory and is in need of improvement.

The main objective of this thesis is to investigate if the currently unstable hover can be improved by using additional information from the hexacopter internal measurement unit (IMU) as well as implement additional control strategies. Some of the key problems have been identified and a summary of the objectives proposed to solve them are listed below:

- Use of nonlinear filtering for sensor fusion of computer vision and IMU data to improve position and velocity estimates.
- Improved altitude control of the hexacopter. A more stable altitude hold will also give better results from the computer vision algorithm.

- Improved horizontal control.

If a sufficiently accurate velocity estimate can be found by sensor fusion of IMU and vision data, this velocity can be integrated to fill the gaps and potentially lower the sampling rate of the vision algorithms, saving computation time. Since both the altitude and horizontal control of the hexacopter depend on their respective velocities, a better controller performance should also be attained if the velocity estimates are more accurate.

1.4. Background

The state of the art in unmanned aerial vehicles (UAV) is in a vast stage of development. Progress is made not only by researchers and corporations but also by dedicated hobbyists involved in open-source projects [3, 4].

Many papers rely either on a GPS signal for outdoor use or a motion capture system to estimate the position of the UAV when flying indoors. This way the UAV is supplied with absolute position and velocity estimates that be used for trajectory following, formation flights etc. Some research project only use on-board sensors which allows for more flexibility even though it greatly increases the complexity of finding sufficiently accurate position estimates at a sufficiently high sampling rate. When all sensors are on board, position estimates become relative and might start to drift and if no offline processing is allowed computational power is often an issue. In [5] all information for navigation is acquired by on-board sensors where position and velocity is estimated by using an optical flow sensor. A simplification has been made though since the UAV always follows a landing pad with a predetermined pattern. In [6] IMU data is fused with data from an on-board laser scanner using a complementary filter but the vision data is processed on an external computer.

To overcome sensor disturbances and inaccuracies correct filtering is often needed when fusing inertial data with vision data or GPS data. For this application the extended Kalman filter is a popular choice [7, 8, 9] but [10, 11, 12, 13] show that a different approach using a Sigma point Kalman filter can outperform the extended Kalman filter. The complementary filter can also be a viable choice because of its theoretical and practical simplicity. It is widely used for fusing gyroscope and accelerometer data of UAVs in order to estimate angles but for full position and velocity estimation of the UAV, the nonlinear Kalman filters seem more popular.

A number of different control algorithms have been implemented to control multirotors. The main focus is often to stabilize the attitude but since that is done by the ArduCopter autopilot it is outside the scope of this thesis. For position control some of the most adopted techniques have been PID control [6, 5, 14, 15] and backstepping control [16, 17]. The PID controller is very dominant and other types of control are usually implemented for research but not for practical use.

1.5. Thesis Outline

Chapter 2 gives an overview of the system by explaining the main hardware, how the hardware is connected and how estimation and control is carried out.

Chapter 3 is about creating a dynamic model of the hexacopter. First the basics of the multirotor structure is presented along with some important notations and explanations of the different coordinate frames that is used. Then the dynamic model is created by Newton-Euler formalism and by using the definition of various forces and torques. Finally some of the equations are replaced by system identification.

Chapter 4 explains how the position and velocity of the hexacopter can be estimated by sensor fusion. For this a detailed description of the unscented Kalman filter and its implementation is given.

Chapter 5 is about control. First the theory and implementation aspects for the PID controller is given and then the models used for control is presented. Cascaded PID controllers are implemented to control the horizontal and vertical position of the hexacopter.

Chapter 6 shows the main structure of the Matlab Simulink model and how scheduling is performed.

Chapter 7 presents the achieved results, both in simulation and on the real system. Here the performance of estimation and control is evaluated and a comparison is made between the extended Kalman filter and the unscented Kalman filter.

Chapter 8 gives a discussion and interpretation of the results in Chapter 7.

Chapter 9 presents the conclusions that have been drawn.

Theory and methods for modeling, estimation and control are presented in their chapters respectively.

2. System overview

This chapter explains the main hardware and system architecture.

2.1. Hardware

ArduCopter 3DR Hexa B The hexacopter is a ArduCopter 3DR Hexa B from 3DRobotics [18]. It is a ready-to-fly kit delivered with an ArduPilot Mega 2.5+ autopilot. Some of the features are fixed-pitch propellers, 850Kv brush-less motors, SimonK Electronic Speed Controllers (ESC), aluminum arms, fiberglass mounting boards, a power distribution board, GPS and a 3DR radio telemetry kit. A picture of the ArduCopter 3DR Hexa B is shown in Fig. 2.1.1 where the blue arms indicate the front of the vehicle.

Figure 2.1.1.: ArduCopter 3DR Hexa B with PandaBoard ES mounted on top

ArduPilot Mega 2.5+ The ArduPilot Mega 2.5+ [15] is an open source autopilot system capable of controlling angular rotations and altitude as well as way point navigation if a GPS signal is available. It features a Atmel ATMEGA2560 chip for processing and a ATMEGA32U-2 chip for USB functions. It includes an Internal Measurement Unit (IMU) with the following sensors:

- InvenSense MPU-6000, 3-axis Gyro / 3-axis Accelerometer.
- Honeywell HMC5883L-TR, 3-axis magnetometer
- Measurement Specialties MS6511-01BA03, Barometric pressure sensor

It also supports data from some external sensors:

- 3DR uBlox LEA-6, GPS
- MaxSonar, Sonar (ultrasonic range finder)

Only the sonar will be used for measuring altitude since the barometer is not accurate enough for indoor use and since it depends on air pressure it can react to if someone opens a door to the room etc. The magnetometer is also ignored because flights are made indoors and metal objects and beams as well as other electronics seem to cause major disturbances. Not using the magnetometer will cause some drift in heading but it tends to be rather small. A GPS signal is unavailable indoors and thus the GPS receiver is ignored. The autopilot is supplied with its own source code.

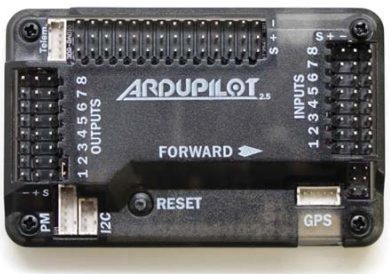

Figure 2.1.2.: ArduPilot Mega 2.5+

PandaBoard ES PandaBoard [19] is a low-cost, open OMAP 4 mobile software development platform. Technical specifications of importance for this thesis is listed below.

- Dual-core ARM Cortex -A9 MPCore 1.2 GHz processor
- 1 GB low power DDR2 RAM
- SD card cage with support for high-speed and high-capacity SD cards

- On-board 10/100 Ethernet
- 1x USB 2.0 High-Speed On-the-go port
- 2x USB 2.0 High-Speed host port
- 802.11 b/g/n (based on WiLink™ 6.0), wireless connection
- Size: 114.3x101.6 mm
- Weight: 81.5g

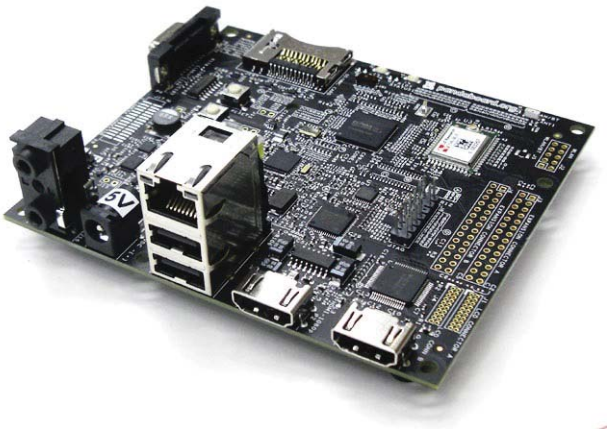

Figure 2.1.3.: PandaBoard ES

Logitech C310 USB Camera Since GPS cannot be used for position estimation indoors a simple web camera is used for position estimation. The camera is mounted underneath the hexacopter facing downwards towards the ground and features 720p resolution, Hi-Speed USB 2.0 and a frame rate of 30 FPS.

Figure 2.1.4.: Logitech C310

Spektrum DX7S Radio Controller

For safety and other practical reasons a radio controller is used to manually disarm the motors and control the hexacopter. By flipping a switch, control is given to the PandaBoard and the hexacopter becomes autonomous. The user can at any given time request control again by flipping back the switch. It features 7 channels and a 2.4GHz DSM spread spectrum telemetry system.

Figure 2.1.5.: Spektrum DX7S Radio Controller

2.2. System architecture

An overview of how the hardware is connected can be seen in Fig. 2.2.1, which is directly taken from [1]. The high level estimation and control is developed in Matlab Simulink and runs on the PandaBoard after automatic code generation into C-code. By using a WiFi-connection it is possible to download the model as well as interact with the running model by the External mode in Simulink. The PandaBoard is supplied with images from the camera via one of the USB ports and it also receives measurements and status data from the ArduCopter via Ethernet. The higher level position controllers on the PandaBoard communicate with the ArduCopter using the same Ethernet connection. The protocol used for the Ethernet connection is UDP. The PandaBoard is mounted on top of the hexacopter and has its own power distribution thanks to the work done by [1].

The control system overview is presented in Fig. 2.2.2. The ArduCopter autopilot will be used for low level control like controlling the angular rotations and setting

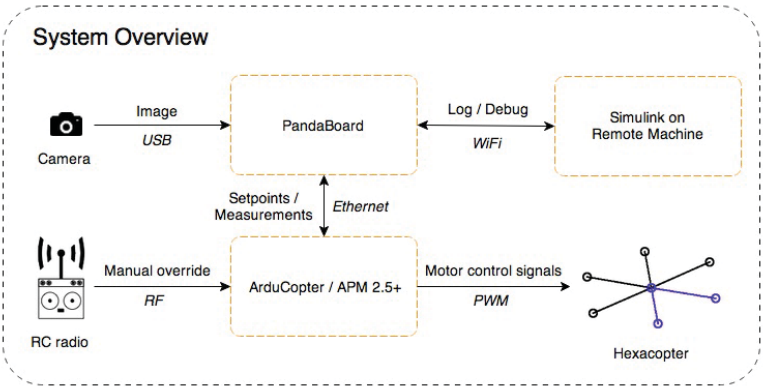

Figure 2.2.1.: System overview

motor outputs and also for providing sensor data used by the higher level estimation and control on the PandaBoard. The IMU, sonar and vision data is fused by using a nonlinear filter (unscented Kalman filter) and the estimates are supplied to the controllers that control the vertical and horizontal position of the hexacopter by sending attitude and throttle setpoints back to the ArduCopter autopilot.

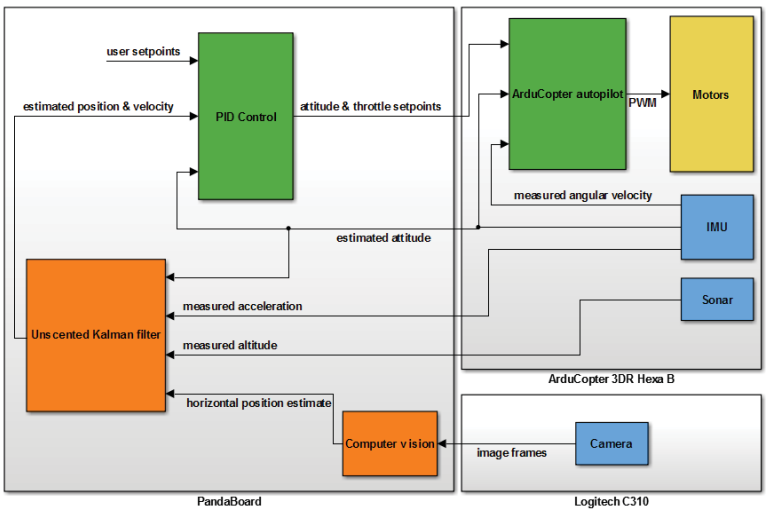

Figure 2.2.2.: Control overview

3. Modeling the system

In this chapter a dynamic model is created to give understanding of the system and to provide equations to be used when designing algorithms for estimation and control.

3.1. Multirotor basics

A hexacopter is a multirotor aerial vehicle with six rotors as represented by Fig. 2.1.1. The multirotor is an under-actuated, dynamically unstable, six degrees of freedom system in very strong need of control. The six degrees of freedom consist of translational and rotational motion in three dimensions where the translational motion is created by changing the direction and magnitude of the upward propeller thrust. For fixed rotor blades (as in this thesis) the rotational motion needed to tilt the thrust vector is accomplished by changing the speed of the propellers individually to create torques around the center of rotation. The sum of the propeller speeds will decide the magnitude of the trust vector. The multirotor motion in two dimensions is depicted by Fig. 3.1.1.

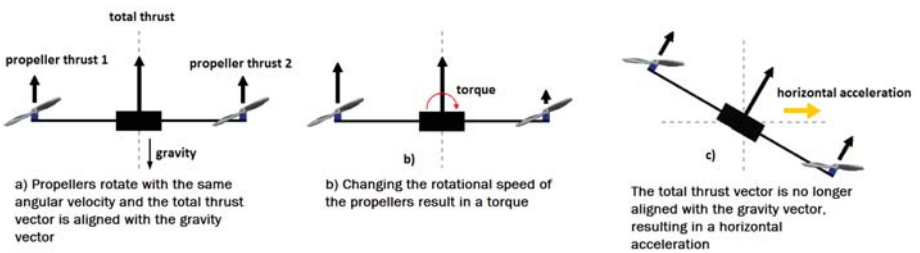

Figure 3.1.1.: Multirotor motion

3.2. Theory

3.2.1. Coordinate frames

To keep track of the hexacopter two different coordinate frames are used to represent the position and orientation in three dimensions, the earth frame and the body frame. The earth frame is a fixed frame used as an unmoving reference. Say for example that the user want to define a path that the hexacopter should follow, then that path would be represented in the earth frame. The body frame axes are aligned with the sensors which is convenient when reading sensor data and controlling the angular orientation (attitude). In this thesis the body frame is defined as having the x-axis pointing forward (indicated by blue arms), y-axis pointing to the left and the z-axis pointing upwards, see Fig. 3.2.1.

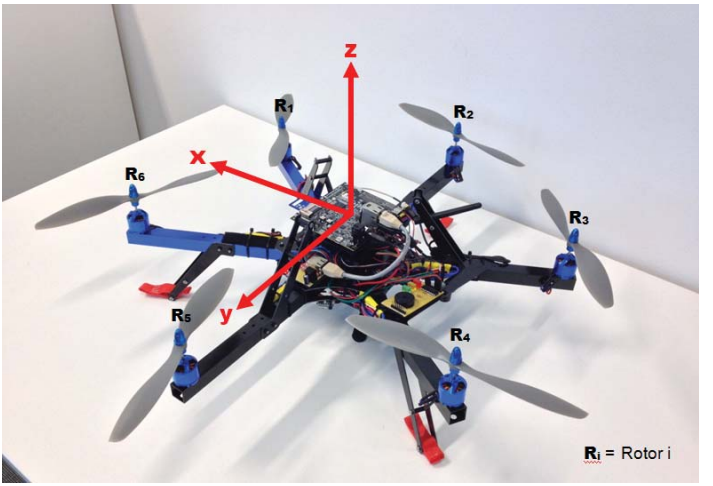

Figure 3.2.1.: Hexacopter body frame and rotor indexing

3.2.2. Angular rotations

The attitude is defined by the orientation of the body frame compared to the earth frame. It represents the rotations about the x-, y- and z-axes, in this case using the right-hand rule, and consists of roll, pitch and yaw, see Fig. 3.2.2. The attitude is controlled by changing the rotor speeds where the rotors are numbered in a clockwise manner with rotor 1 being the front right rotor of the hexacopter, again see Fig. 3.2.1.

Roll is the rotation about the x-axis and is obtained by lowering/increasing the speed of rotor 1, 2 and 3 and at the same time increasing/lowering the speed of rotor 4, 5 and 6. This leads to a torque around the x-axis and thereby an angular acceleration. The notation for the roll angle will be ϕ $[rad/s]$.

Pitch is the rotation about the y-axis and is obtained by lowering/increasing the speed of rotor 1 and 6 and at the same time increasing/lowering the speed of rotor 3 and 4. Since the y-axis coincide with the position of rotor 2 and 5 these do not affect pitch. The notation for pitch angle will be θ $[rad/s]$.

Yaw is the rotation about the z-axis and to control it the fact is used that each propeller cause a torque around the z-axis when it rotates. This torque is directed in the opposite way of the propeller's rotation so if the propeller is rotating clockwise it will cause a counterclockwise rotation about the z-axis. To keep the hexacopter stable the propellers have to be rotated in different directions so that three of the rotors spin clockwise and the remaining three spin counterclockwise. A yaw rotation is then created by decreasing/increasing rotor 1, 3 and 5 and at the same time increasing/decreasing rotor 2, 4 and 6. The notation for the pitch angle will be ψ $[rad/s]$.

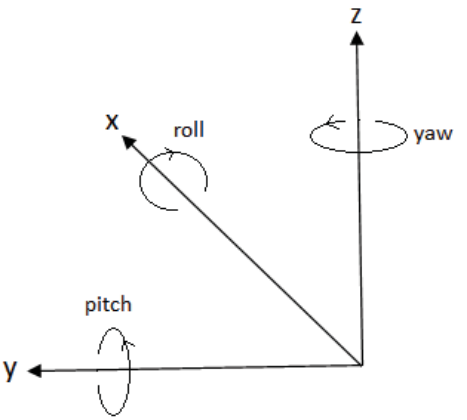

Figure 3.2.2.: Coordinate axes and their relation to roll, pitch and yaw

3.2.3. Rotation matrix

The earth frame and the body frame can be related to each other by a series of rotations [20]:

Yaw rotation

$$R_B^E(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2.1}$$

Pitch rotation

$$R_B^E(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.2.2}$$

Roll rotation

$$R_B^E(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{3.2.3}$$

By performing all three rotations in the presented order the body to earth rotation matrix R_B^E is obtained

$$R_B^E = R_B^E(\psi)R_B^E(\theta)R_B^E(\phi) \tag{3.2.4}$$

$$= \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\theta & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{3.2.5}$$

R_B^E is an orthogonal matrix meaning its inverse equals its transpose so that the earth to body transformation can be done according to

$$(R_B^E)^{-1} = (R_B^E)^T = R_E^B \tag{3.2.6}$$

If for example acceleration is measured with the on board accelerometer the x-, y-, and z-acceleration is obtained in body frame. To get the acceleration in earth frame, the three acceleration measurements are placed in a column vector and multiplied with the rotation matrix, i.e. $accel^{Earth} = R_B^E accel^{Body}$.

3.2.4. Equations of motion

As a base for simulation, estimation and control a model describing the hexacopter and its dynamics is developed. For this the well known Newton-Euler formalism is used to describe the dynamics of a rigid body affected by external forces and torques as explained by [20].

Newton-Euler model Equation 3.2.7 describes a rigid body affected by external forces and torques.

$$\begin{bmatrix} mI_{3x3} & 0_{3x3} \\ 0_{3x3} & I \end{bmatrix} \begin{bmatrix} \dot{V}^B \\ \dot{\omega}^B \end{bmatrix} + \begin{bmatrix} \omega^B \times mV^B \\ \omega^B \times I\omega^B \end{bmatrix} = \begin{bmatrix} F^B \\ \tau^B \end{bmatrix} \tag{3.2.7}$$

Here m $[kg]$ is the mass, I $[Nms^2]$ is the inertia tensor, $V^B = \begin{bmatrix} u & v & w \end{bmatrix}$ $[m/s]$ is the linear velocity in body frame, $\omega^B = \begin{bmatrix} p & q & r \end{bmatrix}$ $[rad/s]$ is the angular velocity in body frame, F^B $[N]$ is the forces affecting the hexacopter in body frame,τ^B $[Nm]$ is the torques affecting the hexacopter in body frame, 0_{3x3} is a zero matrix of size 3 and I_{3x3} is a unit matrix of size 3. By using the fact that a vector cross product can be expressed as a product of a skew-symmetric matrix and a vector according to

$$a \times b = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}, \tag{3.2.8}$$

and by using the approximation that the inertia tensor is diagonal as in

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}, \tag{3.2.9}$$

Equation 3.2.7 can be expanded as described in Equation 3.2.10 - Equation 3.2.11

$$\begin{cases} mI_{3x3}\dot{V^B} + \omega^B \times mV^B = \begin{bmatrix} m\dot{u} \\ m\dot{v} \\ m\dot{w} \end{bmatrix} + \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} mu \\ mv \\ mw \end{bmatrix} = F^B \\ I\dot{\omega}^B + \omega^B \times I\omega^B = \begin{bmatrix} I_{xx}\dot{p} \\ I_{yy}\dot{q} \\ I_{zz}\dot{r} \end{bmatrix} + \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} I_{xx}p \\ I_{yy}q \\ I_{zz}r \end{bmatrix} = \tau^B \end{cases} \tag{3.2.10}$$

$$\begin{cases} \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \begin{bmatrix} \frac{1}{m}F_x^B \\ \frac{1}{m}F_y^B \\ \frac{1}{m}F_z^B \end{bmatrix} \\ \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{I_{yy}-I_{zz}}{I_{xx}}qr \\ \frac{I_{zz}-I_{xx}}{I_{yy}}pr \\ \frac{I_{xx}-I_{yy}}{I_{zz}}pq \end{bmatrix} + \begin{bmatrix} \frac{1}{I_{xx}}\tau_x^B \\ \frac{1}{I_{yy}}\tau_y^B \\ \frac{1}{Izz}\tau_z^B \end{bmatrix} \end{cases} \tag{3.2.11}$$

Equation 3.2.11 is valid for all rigid bodies but to make the system describe a hexacopter the external forces F^B and torques τ^B have to be defined accordingly.

3.2.5. Forces and torques

Here the forces contained in F^B and the torques contained in τ^B will be defined according to [16, 8, 7].

3.2.5.1. Forces

The two main forces come from gravity and the thrust of the rotors but to make the model more realistic rotor drag and air friction is also included.

Gravity

The force of gravity will always point downwards along the Z-axis wrt. the earth frame which in body frame corresponds to

$$F^B_{gravity} = R^B_E \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} = \begin{bmatrix} mg\sin\theta \\ -mg\cos\theta\sin\phi \\ -mg\cos\theta\cos\phi \end{bmatrix} \tag{3.2.12}$$

Thrust

Thrust is the lifting power that makes the hexacopter fly and depends on the sum of the speed of the six propellers. To maintain the total thrust the speed of one propeller is always lowered as much as the speed of another propeller is increased when performing roll, pitch or yaw maneuvers. Since the rotors are fixed their total thrust will always pull upwards along the z-axis wrt. the body frame. In hover conditions this force can be approximated by Equation 3.2.13, where b $[Ns^2]$ is a thrust constant.

$$F^B_{thrust} = b\sum_{i=1}^{6}\Omega_i^2 \tag{3.2.13}$$

Rotor drag

According to [8, 21] there is a drag force acting on the body of any multirotor while flying. This drag force will affect the x and y accelerations wrt. the body frame and can in hover conditions be approximated by Equation 3.2.14, where μ $[kg/s]$ is a constant. A more detailed description of the rotor drag and how it can be used for velocity estimation is given in Sec. 4.2.2.

$$F^B_{drag} = \begin{bmatrix} -\mu u \\ -\mu v \\ 0 \end{bmatrix} \tag{3.2.14}$$

Air resistance

Air resistance is proportional to the squared velocity, size and shape of the object according to

$$F_{air}^{B} = \begin{bmatrix} -\frac{1}{2}CA_x\rho u|u| \\ -\frac{1}{2}CA_y\rho v|v| \\ -\frac{1}{2}CA_z\rho w|w| \end{bmatrix} \tag{3.2.15}$$

Here C is a dimensionless friction constant, A_i $[m^2]$ is the cross-sectional area and ρ $[kg/m^3]$ is the density of air.

3.2.5.2. Torques

In this section the torque resulting from actuator action is very dominant compared to the rest.

Actuator action

As described in Sec. 3.1 increasing/decreasing the speed of the six rotors independently will create torques around the x y z axes and thus creating roll-, pitch- and yaw-rotations. By always decreasing the speed of one rotor as much as increasing the speed of another the total thrust is retained. Since there are six rotors instead of the more traditional four the dynamics become slightly more complicated and some trigonometry needs to be used. Torque is force multiplied by a distance and the rotors will affect the total rotation about a certain axis differently depending on the distance from the center of gravity. Fig. 3.2.3 relates the length and angles of the arms to the relative distance from center of gravity which is the distance from the rotor to the axis of rotation. In Equation 3.2.16 - Equation 3.2.18, $\Omega[rad/s]$ is the propeller speed, l $[m]$ is the arm length, and d $[Nms^2]$ is the drag factor (not related to the rotor drag in 3.2.5.1).

Roll torque

By decreasing Ω_1, Ω_2, Ω_3 and increasing Ω_4, Ω_5, Ω_6 a positive roll moment is produced

$$\tau_{roll} = bl(-\Omega_2^2 + \Omega_5^2 + \frac{1}{2}(-\Omega_1^2 - \Omega_3^2 + \Omega_4^2 + \Omega_6^2)) \tag{3.2.16}$$

Pitch torque

By decreasing Ω_1, Ω_6 and increasing Ω_3, Ω_4 a positive pitch moment is produced

$$\tau_{pitch} = bl\frac{\sqrt{3}}{2}(-\Omega_1^2 + \Omega_3^2 + \Omega_4^2 - \Omega_6^2) \tag{3.2.17}$$

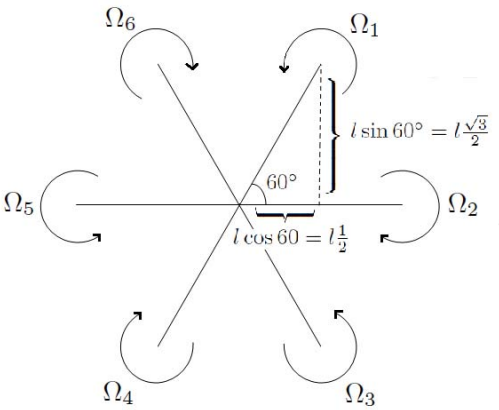

Figure 3.2.3.: Hexacopter rotor distances to center of gravity

Yaw torque

By decreasing Ω_1, Ω_3, Ω_5 and increasing Ω_2, Ω_4, Ω_6 a positive yaw moment is produced

$$\tau_{yaw} = d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2 - \Omega_5^2 + \Omega_6^2) \tag{3.2.18}$$

Note that this not only depends on the definition of the yaw rotation itself but also which rotors that spin clockwise and which rotors that spin counterclockwise. A rotor spinning clockwise will always produce a counterclockwise yaw rotation.

Propeller gyroscopic effect

The rotation of the propellers produces a gyroscopic effect

$$\tau_{gyro} = \begin{bmatrix} -J_r\dot{\theta}\Omega_r \\ J_r\dot{\phi}\Omega_r \\ 0 \end{bmatrix}, \tag{3.2.19}$$

where J_r $[Nms^2]$ is the rotational inertia of the propeller and $\Omega_r = -\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4 - \Omega_5 + \Omega_6$ $[rad/s]$ is the overall propeller speed.

Yaw counter torque

Differences in rotational acceleration of the propellers produces a yaw inertial counter torque

$$\tau_{counter} = \begin{bmatrix} 0 \\ 0 \\ J_r\dot{\Omega}_r \end{bmatrix} \tag{3.2.20}$$

3.2.6. Final system equations

By summing up the expressions derived in Sec. 3.2.4 and Sec. 3.2.5 the final equations of motion are listed in Equation 3.2.21 and their relation to the propeller speed are described by Equation 3.2.22.

$$\begin{cases} \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw & + & g\sin\theta & - & \frac{\mu}{m}u & & & - & \frac{1}{2m}CA_x\rho u|u \\ pw - ru & - & g\cos\theta\sin\phi & - & \frac{\mu}{m}v & & & - & \frac{1}{2m}CA_y\rho v|v| \\ qu - pv & - & g\cos\theta\cos\phi & & & + & \frac{1}{m}F_{thrust} & - & \frac{1}{2m}CA_z\rho w|w| \end{bmatrix} \\ \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{I_{yy}-I_{zz}}{I_{xx}}qr & + & \frac{1}{I_{xx}}\tau_{roll} & - & \frac{1}{I_{xx}}J_r\dot{\theta}\Omega_r & & \\ \frac{I_{zz}-I_{xx}}{I_{yy}}pr & + & \frac{1}{I_{yy}}\tau_{pitch} & + & \frac{1}{I_{yy}}J_r\dot{\phi}\Omega_r & & \\ \frac{I_{xx}-I_{yy}}{I_{zz}}pq & + & \frac{1}{Izz}\tau_{yaw} & & & + & \frac{1}{Izz}J_r\dot{\Omega}_r \end{bmatrix} \end{cases} \tag{3.2.21}$$

$$\begin{cases} F_{thrust} = b\sum_{i=1}^{6}\Omega_i^2 \\ \tau_{roll} = bl(-\Omega_2^2 + \Omega_5^2 + \frac{1}{2}(-\Omega_1^2 - \Omega_3^2 + \Omega_4^2 + \Omega_6^2)) \\ \tau_{pitch} = bl\frac{\sqrt{3}}{2}(-\Omega_1^2 + \Omega_3^2 + \Omega_4^2 - \Omega_6^2) \\ \tau_{yaw} = d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2 - \Omega_5^2 + \Omega_6^2) \\ \Omega_r = -\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4 - \Omega_5 + \Omega_6] \end{cases} \tag{3.2.22}$$

3.3. System identification

Some parts of Equation 3.2.21 and Equation 3.2.22 were modeled by using the system identification toolbox in Matlab. These models were only used for simulation to evaluate the estimation and control algorithms and to tune parameters.

3.3.1. Angular rotations

The ArduCopter 3DR Hexa B comes with an autopilot capable of controlling roll, pitch and yaw in a stable manner and thus no low level control of the attitude has to be developed. However, attitude setpoints are sent from the position controller on the Pandboard to the ArduCopter autopilot which creates a need for a model describing the relation between attitude setpoints and actual attitude. This model was developed by system identification in the previous thesis work [1] using the attitude reference as an input and the measured attitude as output, thus including both the physics and the controller. A transfer function consisting of 2 poles and 1 zero with a delay proved to be sufficient and provided a fit of about 80% when testing the model with validation data.

3.3.2. Thrust

Instead of trying to model the electronic speed controllers (ESC) and the motors of the hexacopter a more heuristic approach was taken to determine the thrust for a given pulse width modulation (PWM) signal. The hexacopter was attached to a ABB robot with a pressure sensor in the robotics lab at LTH, see Fig. 3.3.1.

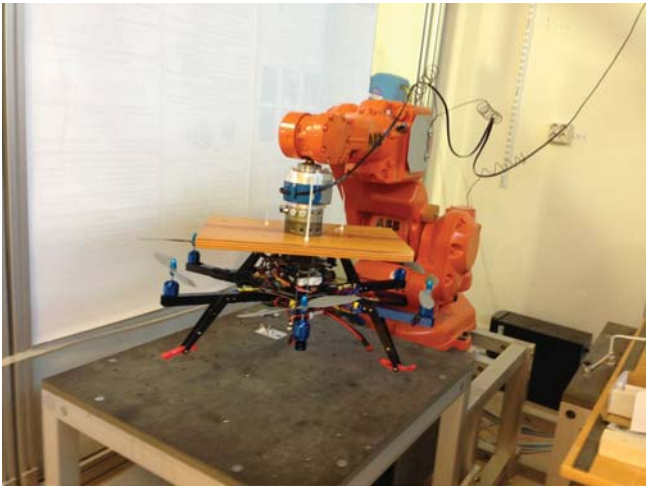

Figure 3.3.1.: Experimental setup for identification of the relation between PWM and thrust

A series of steps were performed, raising the PWM signal and then comparing it to the pressure reading from the sensor on the robot. From this a first order transfer function from PWM to thrust was identified according to

$$G(s) = K_{BL}\frac{1}{\frac{1}{\tau}s+1} = K_{BL}\frac{1}{\frac{1}{5.085}s+1}, \tag{3.3.1}$$

where K_{BL} is a steady-state gain depending on the current battery level. K_{BL} was identified by plotting PWM against steady state thrust and then using a simple curve fit.

An evaluation was made by feeding a logged PWM signal from a real flight to the model and then comparing its output to the logged accelerometer data, see Fig. 3.3.2.

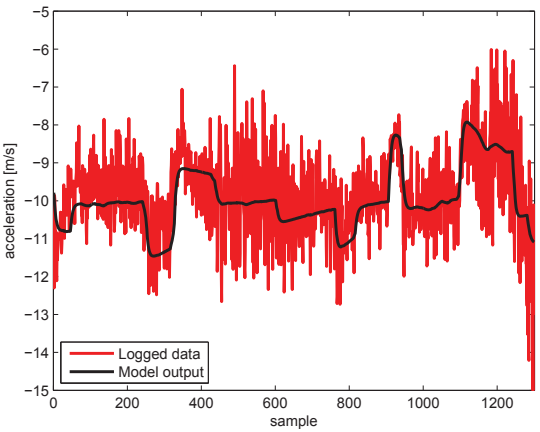

Figure 3.3.2.: Thrust model validation

4. Estimation

One of the true cornerstones for any autonomous UAV flight is to have correct information about its orientation, position and velocity. If missions take place in an outdoor environment, GPS is almost always the solution to get position estimates of sufficient accuracy. When flying indoors, a motion capture system can be used to get good information about orientation, position and velocity. But a good motion capture system is very expensive and it is only viable for tests at a predetermined location. If all sensors are carried on-board the UAV becomes much more flexible. One viable option is to use a laser scanner, even though the information often is in need of processing on an external computer.

In [1] a simpler approach was made by using a USB camera to get visual information of the ground below the multirotor. By using an algorithm called template matching, image frames are compared and the pixel displacement is used together with attitude and altitude information to estimate movement. The algorithm works in simulation, looking at a photo that was taken on a play mat with distinct patterns, but shows a rather weak performance in reality even though the same play mat is placed on the ground below.

So the question is how to improve the position estimation and at the same time get good velocity estimates to improve control? In this thesis the needed improvement is searched for in the additional information provided by the IMU. The accelerometers can be integrated to get velocity and the velocity can be integrated to get position. With the attitude information provided, transformations can be made between the body frame and the earth frame and the impact of gravity can be compensated for. But to do this great care has to be taken because of the imperfection of the sensors. The accelerometers are very noisy from the vibration caused by the rotors and from disturbances due to other electronics, they are biased and also subject to slow drift. Attitude information is obtained by integrating gyroscopic measurements that is corrected by the accelerometers and here again noise, bias and drift is an issue. The errors in the sensor measurements will magnify extremely fast for each integration making position and velocity estimation based on pure IMU data very hard if not impossible with low cost sensors. The IMU data will not be used by its own but instead it will be used to improve the estimates provided by the vision system and the sonar. To address the imperfections of the sensors good filtering is still very important.

To get the best result possible all information should be used. The available measurements for estimation are listed below.

- Acceleration measurements from the IMU 3-axis accelerometer
- Attitude estimates from the ArduCopter autopilot (the autopilot fuses gyro and accelerometer data from the IMU to estimate attitude)
- Altitude measurements from the sonar.
- Horizontal position estimates from the computer vision algorithm

The listed information will be fused together in a nonlinear unscented Kalman filter. The theory will be explained in the next section and after that the implementation details are presented.

4.1. Theory

To fuse the available data an Unscented Kalman filter (UKF) is used. The UKF is a recursive minimum mean-square error estimator based on the Kalman filter [22]. First a brief explanation is given of the classical nonlinear version of the Kalman filter, namely the extended Kalman filter (EKF). Then the so-called Sigma-point Kalman filter approach is explained followed by the details of the unscented transform and finally the filter used in this thesis is presented which implements the square-root version of the UKF.

4.1.1. Nonlinear model

The hexacopter will be modeled as a nonlinear system observed by measurements according to

$$\begin{cases} \mathbf{x}_{k+1} &= \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{v}_k \\ \mathbf{y}_k &= \mathbf{h}(\mathbf{x}_k) + \mathbf{n}_k \end{cases}, \tag{4.1.1}$$

where $\mathbf{x}_k$ is the current state, $\mathbf{u}_k$ is an external input (control signal), $\mathbf{y}_k$ is the observed measurement,$\mathbf{v}_k$ is the process noise that also drives the dynamic system, $\mathbf{n}_k$ is measurement noise and $\mathbf{f}(\cdot)$ and $\mathbf{h}(\cdot)$ represent the dynamic nonlinear process and measurement model respectively. In this thesis $\mathbf{v}_k$ and $\mathbf{n}_k$ are assumed to be additive zero mean Gaussian noise with covariance $\mathbf{Q}_k$ and $\mathbf{R}_k$ respectively.

4.1.2. The extended Kalman filter

By linearization the EKF [23] can, unlike the Kalman filter, use a nonlinear model in its estimation according to Equation 4.1.1. Still some authors [10] would not like to call it a proper extension of the Kalman filter. The Kalman filter presents an optimal

solution if the model is perfect, the noise is white and its covariance is known. The EKF is more of a way to try and use the Kalman filter by approximating a nonlinear system with a linear one. Still it has proven successful in many applications and has been used extensively for many years in both state estimation and parameter estimation. The EKF algorithm is given in Appendix A.

Since the EKF uses the Jacobian, which is a first order Taylor expansion, when calculating the mean and covariance some linearization errors cannot be avoided. A key weakness is also that the linearization is only made around a single point which does not take into account that the state is a random variable. This means that the mean of the state is not estimated as $\overline{\mathbf{x}}_{k+1} = E\left[\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k)\right]$ but instead as $\overline{\mathbf{x}}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k)$. The expectation is also not included when calculating $\overline{\mathbf{y}}_k$ or the estimated state covariance. The magnitude of these errors depend on how nonlinear the system is and will give varying results thereafter. High nonlinearities will result in larger errors and sometimes divergence.

4.1.3. Sigma-point Kalman filters

A different approach that has proven successful in a number of applications is the family of Sigma-point Kalman filters [10][11]. The Sigma-point Kalman filter utilizes a deterministic sampling approach to determine the mean and covariance of the state, which is still assumed to be a Gaussian random variable. The idea is to create a set of points to represent a discrete approximation of the state distribution, fully capturing the first and second moment (mean and covariance) and then propagate it through the nonlinear model. The distribution of the transformed points will then be an estimate of the nonlinear transformation of the original distribution. This approach, according to [10], follows from the intuition that

> "It should be easier to approximate a Gaussian distribution than it is to approximate an arbitrary nonlinear function".

Meaning that it should be easier to use this sampling approach, looking at how the Sigma-points have been transformed, then to try and understand and model how the nonlinear function will change the statistics of its input.

One can summarize the most important steps as the following:

1. Create a set consisting of a minimal number of carefully chosen Sigma-points that fully represent the mean and covariance of the state.
2. Propagate the Sigma-points through the nonlinear model.
3. Calculate the distribution of the transformed Sigma-points, usually by a weighted mean and covariance.

This will capture any nonlinearities up to the second moment without error, sometimes higher depending on the distribution of the state. Higher order nonlinearities can also be captured at the cost of more sigma points. This means that the filter is

not subject to the flaws of the EKF that was previously mentioned, that it is derivative free and that it also can use a model according to Equation 4.1.2 where a more sophisticated noise model can be included compared to Equation 4.1.1. Here the noise is not just assumed to be additive but can instead be included in the nonlinear state and measurement models.

$$\begin{cases} \mathbf{x}_{k+1} &= \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k) \\ \mathbf{y}_k &= \mathbf{h}(\mathbf{x}_k, \mathbf{n}_k) \end{cases} \tag{4.1.2}$$

4.1.4. The unscented transform

A key question is how to chose the Sigma-points and one of the most popular and adopted ways of choosing them is the unscented transform presented by [24]. Assume that the state vector $\mathbf{x}$ is of dimension N and that it is a random variable with mean $\overline{\mathbf{x}}$ and covariance $\mathbf{P}_x$. The random variable $\mathbf{y}$ with mean $\overline{y}$ and covariance $\mathbf{P}_y$ is the result of $\mathbf{y} = \mathbf{f}(x)$, where $\mathbf{f}(\cdot)$ is a nonlinear function. To estimate the statistics of $\mathbf{y}$ the Sigma-Points $\mathbf{S} = \{\mathbf{w}_i, \chi_i, i = 1, ..., 2N+1\}$ are chosen as described by Equation 4.1.3 and Equation4.1.4. The notation $\mathbf{A}_i$ represents the ith column of matrix $\mathbf{A}$, the first index of a vector and the first column of a matrix is defined as i=1.

$$\begin{cases} \chi_i = \overline{x} & i = 1 \\ \chi_i = \overline{x} + (\sqrt{(N+\lambda)\mathbf{P}_x})_{i-1} & i = 2, ..., N+1 \\ \chi_i = \overline{x} - (\sqrt{(N+\lambda)\mathbf{P}_x})_{i-N-1} & i = N+2, ..., 2N+1 \end{cases} \tag{4.1.3}$$

$$\begin{cases} w_1^{(m)} = \frac{\lambda}{N+\lambda} & i = 1 \\ w_1^{(c)} = \frac{\lambda}{N+\lambda} + (1 - \alpha^2 + \beta) & i = 1 \\ w_i^{(m)} = w_i^{(c)} = \frac{1}{2(N+\lambda)} & i = 2, ..., 2N+1 \\ \lambda = \alpha^2(N+\kappa) - N \end{cases} \tag{4.1.4}$$

Here w_i is a set of weights that depend on the scaling parameters α, β and κ. α is used to control the spread of the Sigma-points to avoid sampling of non-local effects and should be constrained as $0 < \alpha < 1$, β should be non-negative and can be used for dealing with higher order moments of the statistics, for a Gaussian distribution $\beta = \sqrt{2}$ has been proven optimal. κ is also non-negative and it is used to guarantee that the covariance matrix is positive definite, usually $\kappa = 0$.

S is now used instead of x as an input to the nonlinear function:

$$\mathbf{Y}_i = \mathbf{f}(\mathbf{S}_i)\ i = 1, ...2N+1 \tag{4.1.5}$$

Finally the transformed distribution can be estimated by using the standard formulas for mean and covariance together with the weights defined in Equation 4.1.4:

$$\overline{y} = \sum_{i=1}^{2N+1} w_i \mathbf{Y}_i \tag{4.1.6}$$

$$\mathbf{P}_y = \sum_{i=1}^{2N+1} w_i (\mathbf{Y}_i - \overline{y})(\mathbf{Y}_i - \overline{y})^T \tag{4.1.7}$$

$$\mathbf{P}_{xy} = \sum_{i=1}^{2N+1} w_i (\chi_i - \overline{x})(\mathbf{Y}_i - \overline{y})^T \tag{4.1.8}$$

4.1.5. Square root unscented Kalman filter implementation

One weakness of the standard unscented Kalman filter algorithm is the updating and calculation of the square-root of the state covariance matrix $\mathbf{P}$. In some scenarios $\mathbf{P}$ might not become positive definite and it will lead to divergence. To take care of this problem an alternative form of the unscented Kalman filter is introduced by [11] called the square-root unscented Kalman filter (SRUKF). This filter operates with $\mathbf{P}$ in a Cholesky-factored form all the time and uses some additional linear algebra. The result is a much more numerically robust filter, still with the same estimation accuracy and sometimes even a better computational cost depending on the model and its size. Therefore this will be the filter of choice. A complex noise model will not be used but instead it is assumed that the noise affecting the process and measurements is additive and follows the one defined in Equation 4.1.1. For additive noise [11] states that the computational complexity of the filter is $\mathcal{O}(N^3)$ where N is the number of states which is the same order as the EKF.

This filter algorithm is summarized below. For understanding purposes some comparisons are made to the standard UKF since it is more intuitive and the linear algebra used in the SRUKF can make it hard to understand what is really happening.

The square root additive noise unscented Kalman filter algorithm

- Initialization, $k = 0$

$$\hat{\mathbf{x}}_0 = E[\mathbf{x}_0],\ \mathbf{S}_{x_0} = chol\{E[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^T]\} \tag{4.1.9}$$

$$\mathbf{S}_Q = chol\{\mathbf{Q}\},\ \mathbf{S}_R = chol\{\mathbf{R}\} \tag{4.1.10}$$

$\mathbf{R}_{chol} = chol\{\mathbf{A}\}$ represents the Cholesky factorization of the symmetric and positive definite matrix $\mathbf{A}$, which is a numerically efficient way of calculating the square root satisfying $\mathbf{R}_{chol}^T\mathbf{R}_{chol} = \mathbf{A}$.

- For $k = 1, ..., \infty$

  1. Calculate new Sigma-points

$$\begin{cases} \chi_{i,k-1} = \hat{\mathbf{x}}_{k-1} & i = 1 \\ \chi_{i,k-1} = \hat{\mathbf{x}}_{k-1} + \gamma\mathbf{S}_{i-1,x_{k-1}} & i = 2, ..., N+1 \\ \chi_{i,k-1} = \hat{\mathbf{x}}_{k-1} - \gamma\mathbf{S}_{i-N-1,x_{k-1}} & i = N+2, ..., 2N+1 \end{cases} \tag{4.1.11}$$

  This follows the definition of the unscented transform presented in Equation 4.1.3. Here $\gamma = \sqrt{N+\lambda}$ is a scaling parameter, N is the number of states and λ was defined in Equation 4.1.4. The notation $\mathbf{S}_{i,x_{k-1}}$ is used to represent the ith column of $\mathbf{S}_{x_{k-1}}$

  2. Predict

  In the prediction phase the Sigma-points are propagated through the nonlinear process model and these transformed Sigma-points are then used to predict the state and its covariance. The predicted states are then used to form a new set of Sigma-points which are propagated through the nonlinear measurement model. This second set of transformed Sigma-points are used to calculate the predicted output. To keep the covariances in a square root form and at the same time be numerically efficient and robust some linear algebra will be used.

  Propagate the Sigma-points through the nonlinear process model

$$\chi^-_{i,k|k-1} = \mathbf{f}(\chi_{i,k-1}, \mathbf{u}_{k-1}) \tag{4.1.12}$$

  Use the transformed Sigma-points to predict the state, w_i was defined in Equation 4.1.4.

$$\hat{\mathbf{x}}^-_k = \sum_{i=1}^{2N+1} w_i^{(m)}\chi_{i,k|k-1} \tag{4.1.13}$$

  Predict the covariance of the estimate

$$\mathbf{S}^-_{x_k} = qr\{\begin{bmatrix} \sqrt{w_1^{(c)}}(\chi_{2:2N+1,k|k-1} - \hat{\mathbf{x}}^-_k) & \mathbf{S}_Q \end{bmatrix}\} \tag{4.1.14}$$

$$\mathbf{S}_{x_k}^- = cholupdate\{\mathbf{S}_{x_k}^-, \chi_{1,k|k-1}^- - \hat{\mathbf{x}}_k^-, w_0^{(c)}\} \tag{4.1.15}$$

$qr\{\mathbf{A}\}$ represents the upper triangular part of $\mathbf{R}$ from performing a QR factorization of matrix $\mathbf{A}$. A QR factorization decomposes $\mathbf{A}$ into an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$, $\mathbf{A} = \mathbf{QR} = \mathbf{Q}\begin{bmatrix} \mathbf{R}_1 \\ 0 \end{bmatrix}$. Here $\mathbf{R}_1$ equals the upper triangular factor of the Cholesky factorization of $\mathbf{A}^T\mathbf{A}$. Note that $\mathbf{Q}$ and $\mathbf{R}$ are only used here to describe the QR factorization and have nothing to do with the process noise or measurement noise covariance matrices. $\chi_{2:2N+1,k|k-1} - \hat{\mathbf{x}}_k^-$ means that $\hat{\mathbf{x}}_k^-$ is subtracted from each column of the matrix consisting of column 2,...,2N+1 of $\chi_{k|k-1}$.
$cholupdate\{\mathbf{R}, \mathbf{x}, \pm\alpha\}$ represents the rank 1 update to Cholesky factorization. If $\mathbf{R} = chol\{\mathbf{A}\}$ is the original Cholesky factorization of $\mathbf{A}$, then $cholupdate\{\mathbf{R}, \mathbf{x}, \pm\alpha\}$ will return the Cholesky factorization of $\mathbf{A} \pm \alpha\mathbf{x}\mathbf{x}^T$ where x is a column vector of the same size as the columns of $\mathbf{A}$. If $\mathbf{x}$ is a matrix then an update for each column of $\mathbf{x}$ will be made.
The equivalence of Equation 4.1.14 and Equation 4.1.15 in the original UKF is presented in Equation 4.1.16. The difference between the UKF and the SRUKF is that in the SRUKF the covariance matrix is updated in square root form in a numerically efficient way.

$$\mathbf{P}_{x_k}^- = \sum_{i=1}^{2N+1} w_i^{(c)} (\chi_{i,k|k-1}^- - \hat{\mathbf{x}}_k^-)(\chi_{i,k|k-1}^- - \hat{\mathbf{x}}_k^-)^T + \mathbf{Q} \tag{4.1.16}$$

Form a new set of Sigma-points using the predicted state and estimate covariance

$$\chi_{k|k-1} = \begin{bmatrix} \hat{\mathbf{x}}_k^- & \hat{\mathbf{x}}_k^- + \gamma\mathbf{S}_{x_k}^- & \hat{\mathbf{x}}_k^- - \gamma\mathbf{S}_{x_k}^- \end{bmatrix} \tag{4.1.17}$$

Transform the new Sigma-points using the nonlinear measurement model

$$\mathbf{Y}_{i,k|k-1} = \mathbf{h}(\chi_{i,k|k-1}),\ i = 1, ..., 2N + 1 \tag{4.1.18}$$

Use the transformed Sigma-points to predict the measurement

$$\hat{\mathbf{y}}_k^- = \sum_{i=1}^{2N+1} w_i^{(m)} \mathbf{Y}_{i,k|k-1} \tag{4.1.19}$$

3. Update

In the update phase the Kalman gain $\mathbf{K}$ is calculated and used to weigh the predicted output with the measurement and then update the state and its covariance. Again the linear algebra methods introduced in the predict phase will be used.

Calculate the covariance of the measurement residual

$$\mathbf{S}_{\tilde{y}_k} = qr\{\begin{bmatrix} \sqrt{w_2^{(c)}}(\mathbf{Y}_{2:2N+1,k|k-1} - \hat{\mathbf{y}}_k^-) & \mathbf{S}_R \end{bmatrix}\} \tag{4.1.20}$$

$$\mathbf{S}_{\tilde{y}_k} = cholupdate\{\mathbf{S}_{\tilde{y}_k}, \mathbf{Y}_{1,k|k-1} - \hat{\mathbf{y}}_k^-, w_1^{(c)}\} \tag{4.1.21}$$

Note that $\mathbf{S}_{\tilde{y}_k}$ is the covariance of $\tilde{\mathbf{y}}_k = \mathbf{y}_k - \hat{\mathbf{y}}_k^-$. Equation 4.1.20 and Equation 4.1.21 is the square root version of Equation 4.1.7. Since additive noise is assumed, $\mathbf{S}_R$ is also included.

Calculate the cross-covariance between the predicted state and the predicted measurement

$$\mathbf{P}_{x_k y_k} = \sum_{i=1}^{2N+1} w_i^{(c)} (\chi_{i,k|k-1} - \hat{\mathbf{x}}_k^-)(\mathbf{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-)^T \tag{4.1.22}$$

Calculate the Kalman gain

$$\mathbf{K}_k = (\mathbf{P}_{x_k y_k} / \mathbf{S}_{\tilde{y}_k}^T) / \mathbf{S}_{\tilde{y}_k} \tag{4.1.23}$$

where the "/" operator represents efficient back-substitution and is used instead of a matrix inversion which would be more costly and less numerically stable. Since the covariance of the measurement residual is in square root form, two back-substitutions have to be made.

Weigh the predicted output against the real measurement $\mathbf{y}_k$ by using

the Kalman gain and update the state.

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{y}_k - \hat{\mathbf{y}}_k^-) \tag{4.1.24}$$

Update the estimate covariance

$$\mathbf{U} = \mathbf{K}_k \mathbf{S}_{\tilde{y}_k} \tag{4.1.25}$$

$$\mathbf{S}_{x_k} = cholupdate\{\mathbf{S}_{x_k}^-, \mathbf{U}, -1\} \tag{4.1.26}$$

Again linear algebra is used to make the final update of the state covariance in a square root form. In the standard UKF Equation 4.1.25 and Equation 4.1.26 is replaced by Equation 4.1.27.

$$\mathbf{P}_{x_k} = \mathbf{P}_{x_k^-} - \mathbf{K}_k \mathbf{P}_{\tilde{y}_k} \mathbf{K}_k^T \tag{4.1.27}$$

4.2. Method

In Sec. 4.1.5 the SRUKF was explained but to use it a filter model describing the system has to be supplied. The model presented in Equation 4.1.1 will be used but first it has to be discretized. This is accomplished by first updating the states using the continuous-time process model and then integrating them using the filter step size. This way the continuous-time differential equations from Chapter 3 can be used while measurements are treated as discrete events. In simulation integration by forward Euler proved to give equal results as integration by the fourth order Runge-Kutta method so forward Euler will be chosen because of its low complexity. The process model used for implementation can now be described by

$$\begin{cases} \mathbf{x}_{k+1} & = \mathbf{x}_k + h\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{v}_k \\ \mathbf{y}_k & = \mathbf{h}(\mathbf{x}_k) + \mathbf{n}_k \end{cases} \tag{4.2.1}$$

where h $[s]$ is the step size.

4.2.1. Process model

The process model will be a simplified version of Equation 3.2.21 and follows an approach inspired by [25, 26]. The states consist of position in earth frame, velocity in body frame, velocity bias, drag force coefficient and thrust from the rotors according to

$$\mathbf{x} = [x^E, y^E, z^E, u, v, w, b_u, b_v, b_w, \mu, F_{thrust}] \tag{4.2.2}$$

The reason for having the velocity bias states is that the velocity states can become biased because of imperfections in the angular measurements. The states $b_u, b_v, b_w, \mu, F_{thrust}$ are modeled as a random walk and the earth frame position estimates are found by multiplying the rotation matrix $\mathbf{R}_B^E$ with the body frame velocity estimates and then integrating. The final model for estimation can now be described by

$$\begin{bmatrix} \dot{x}^E \\ \dot{y}^E \\ \dot{z}^E \\ \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{b}_u \\ \dot{b}_v \\ \dot{b}_w \\ \dot{\mu} \\ \dot{F}_{thrust} \end{bmatrix} = \begin{bmatrix} (c\psi c\theta)u & + & (c\psi s\theta s\phi - s\psi c\phi)v & + & (c\psi s\theta c\phi + s\psi s\phi)w \\ (s\psi c\theta)u & + & (s\psi s\theta s\phi + c\psi c\theta)v & + & (s\psi s\theta c\phi - c\psi s\phi)w \\ -(s\theta)u & + & (c\theta s\phi)v & + & (c\theta c\phi)w \\ gs\theta & - & \frac{\mu}{m}u & - & b_u \\ -gc\theta s\phi & - & \frac{\mu}{m}v & - & b_v \\ -gc\theta c\phi & + & \frac{1}{m}F_{thrust} & - & b_w \\ 0 & & & & \\ 0 & & & & \\ 0 & & & & \\ 0 & & & & \\ 0 & & & & \end{bmatrix} \tag{4.2.3}$$

4.2.2. Measurement model

The available measurements come from the vision system, sonar, accelerometers and attitude estimates from the autopilot. This gives the measurement vector presented in Equation 4.2.4. Since the ArduCopter autopilot already fuses gyro data with accelerometer data to produce attitude estimates they are not used in the measurement function but instead they are included in the filter separately to save computation time.

$$\mathbf{y} = [x_{vision}, y_{vision}, z_{sonar}, a_x, a_y, a_z] \tag{4.2.4}$$

To relate the measurements to the states in Equation 4.2.2 the following measurement model is used

$$\begin{bmatrix} x_{vision} \\ y_{vision} \\ z_{sonar} \\ a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} x^E \\ y^E \\ z^E \\ -\frac{\mu}{m}u \\ -\frac{\mu}{m}v \\ F_{thrust} \end{bmatrix} \tag{4.2.5}$$

A common assumption is that the x- and y-accelerometers give a measurement of gravity. This is because accelerometers measure the difference in linear acceleration between the reference frame of the accelerometer (body frame) and the earth frames gravitational vector. But that is only true if the accelerometer is stationary and not affected by other forces. The reason for having the μ variable to affect the x- and y-accelerometer readings is that recent research [25, 21] indicate that assuming that the x and y accelerometers of a multirotor would measure gravity is not totally correct. The explanation is that the multirotor is affected by an additional force identified as rotor drag. Experiments show that that the x- and y-accelerometer readings instead can give a direct measurement of the linear velocities and that the μ variable is observable. Fig. 4.2.1 is presented in [8]. Here the red signal is the measured acceleration and the black signal is the estimated acceleration by using $a_x^B = \frac{\mu}{m}u$, where u is measured by a motion capture system.

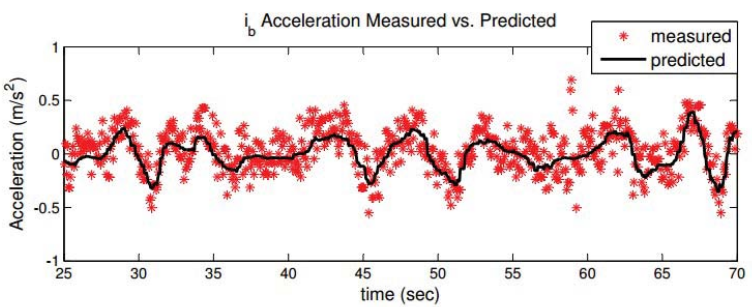

Figure 4.2.1.: Validation of body acceleration measurement model, taken from [8]

Measurements arrive to the filter at different rates so for every iteration the measurement model and measurement covariance matrix is adjusted so that it only contains the elements that correspond to the available measurements.

4.2.3. Noise model

Process noise The process noise covariance matrix $\mathbf{Q}$ can be seen as the model uncertainty and and it is also what drives the system forward. For the states modeled as a random walk this will be the only term in their update model. In the filter $\mathbf{Q}_k$ will be a diagonal matrix mainly to be used as a tuning parameter. Noise is assumed to enter the system at the highest order differential equation so the process noise values of position and velocity is set to a low value. The rest of the states are random-walk states and here the process noise is used to control how fast they can change. To transform $\mathbf{Q}$ into discrete time the approximation described by Equation 4.2.6 is used, where h is the sampling time and Φ is the discrete state transition matrix of the system.

$$Q_k = \int_0^h \Phi(\tau) Q \Phi^T(\tau) d\tau \approx Qh \tag{4.2.6}$$

This first order approximation ignores the terms in Q_k that would have been multiplied with h^N, $\{N \in \mathbb{Z} \mid N \geq 2\}$.

Measurement noise The measurement noise $\mathbf{R}$ represents the uncertainty of the measurements and can be estimated by looking at the data produced by the given sensors. Data is logged and then the measurement noise covariance is calculated using off-line analysis. The accelerometer and sonar data are high-pass filtered in Matlabs System Identification Toolbox to distinguish the noise from the wanted signal and then the noise covariances and cross-covariances are determined. Since no ground truth information is available the covariance of the vision estimates will be hand-tuned without any initial guess. The cross-covariances that involve the vision measurements are set to zero.

4.2.4. Bias removal of acceleration and angular measurements

The measurements of acceleration and attitude from the ArduCopter autopilot are biased which needs to be compensated for. This can be done in the SRUKF but it will increase the computational complexity of the filter and as the biases only tend to drift slowly within a small range it is removed with a simple low-pass filter according to Equation 4.2.7 where α is a filter constant with a low value.

$$bias_k = bias_{k-1} + \alpha * (measurement_k - bias_{k-1}) \tag{4.2.7}$$

5. Control

The hexacopter position is controlled by sending attitude and thrust setpoints to the ArduCopter autopilot. These setpoints are the output of cascaded PID controllers on the PandaBoard whose structure will be explained in this chapter.

5.1. Theory

5.1.1. PID control

Proportional Integral Derivative (PID) control is explained in [27] and is one of the most fundamental types of control and also the most frequently implemented in industry. The great strengths of the PID controller are its simple structure and low requirements on the system model. The purpose of the PID controller is to minimize the current error $e(t) = r(t) - y(t)$, where $r(t)$ is a desired value, called reference or setpoint, and $y(t)$ is a measured value from the process, see Fig. 5.1.1. Since the output of the process is used by the controller to calculate a control signal that is fed back into the process a closed loop is formed.

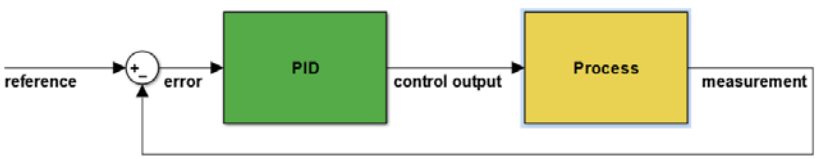

Figure 5.1.1.: PID control, overview

The controller output consists of three terms that give it its name:

Proportional term, P

This term is a direct scaling of the error,

$$P = K_p e(t) \tag{5.1.1}$$

where the proportional gain, K_p, is a constant. If K_p is high then the control signal will be large if the error is large, making the controller more responsive at the cost

of lowered stability and potential overshoot. Since the proportional term is a scaling of the error it will become smaller and smaller when the error goes to zero, leaving a steady-state error that depends on the size of K_p. A high value of K_p will give a smaller steady-state error, in theory, an infinite gain would leave a steady-state error of zero. In many implementations, the proportional part has the largest influence on the control signal.

Integral term, I

The I-term depends on the accumulated differences of old errors,

$$I = K_i \int_0^t e(\tau) d\tau \tag{5.1.2}$$

where the integral gain, K_i, is a constant. By integrating the error signal the output will reach the reference faster and the previously mentioned steady-state error is removed. The downside is that the integral part might increase too fast causing the control signal to become larger then needed to hold the error at zero and since the I-term only can become smaller if the error is negative, it will cause overshoot.

Derivative term, D

The D-term is proportional to the derivative of the error,

$$D = K_D \frac{d}{dt} e(t) \tag{5.1.3}$$

where the derivative gain, K_D, is a constant. By looking at the derivative it is possible to predict the future to punish fast changes and by that prevent overshoot and induce stability into the closed loop system. Since the derivative of the error is used the D-term is prone to amplifying noise and can therefore be dangerous to use without the correct filtering.

Total output

The total output of the PID controller is

$$u(t) = \underbrace{K_p e(t)}_{P} + \underbrace{K_i \textstyle\int_0^t e(\tau) d\tau}_{I} + \underbrace{K_D \frac{d}{dt} e(t)}_{D} \tag{5.1.4}$$

All three terms of the PID controller do not have to be used, any combination of the P-, I- and D-term is possible.

5.1.1.1. Implementation aspects

The basics of the PID controller have been explained but to use it in reality some extra considerations should be made. The most important issues and their solutions are described by [27] and are listed below.

Saturation In most cases the absolute value of the control signal $|u(t)|$ is limited either by the actuators of the system or because a higher value might be considered dangerous both for the equipment and for stability. To cope with this the control signal is saturated to fit within a predetermined bound.

Integral windup If the control signal is saturated the integral term will rise (wind up) and cause a large overshoot when the control signal is not saturated anymore. This is of course an unwanted effect and can be solved in a few different ways. In this thesis back-calculation is used which means that that the difference between the control signal and the saturated control signal is multiplied with a constant gain and then subtracted from the integral term to lower it until the control signal is no longer in need of saturation.

Derivative filtering To prevent the derivative term from amplifying noise a first order low-pass filter is used. This will only be applied for the derivatives that are not supplied from the SRUKF.

Setpoint scaling If a reference is changed as a step it will have bad effects on the derivative term. If the reference is constant between the steps the derivative term can be calculated according to

$$\frac{d}{dt}e(t) = \frac{d}{dt}r(t) - \frac{d}{dt}y(t) = -\frac{d}{dt}y(t) \tag{5.1.5}$$

5.1.1.2. Discrete form

The controller will run on a discrete system with sampling time h and thus a discrete form needs to be used. This is accomplished by first moving Equation 5.1.4 into the Laplace domain, see Equation 5.1.6. Note that the derivative term has been modified to include the low-pass filtering.

$$U(s) = (K_p + \frac{1}{s}K_i + K_D\frac{N}{1 + N\frac{1}{s}})E(s) \tag{5.1.6}$$

After that the forward-Euler approximation is used for the integral term, see Equation 5.1.7, and the backward-Euler approximation for the derivative term, see Equation 5.1.8.

$$s \approx \frac{z-1}{h} \tag{5.1.7}$$

$$s \approx \frac{z-1}{hz} \tag{5.1.8}$$

The discrete controller is described by

$$U(z) = K_p E(z) + K_i h \frac{1}{z-1} E(z) + K_D \frac{N}{1 + N h \frac{z}{z-1}} E(z) \tag{5.1.9}$$

5.2. Method

5.2.1. Altitude control

The goal of the altitude controller is to keep the hexacopter at a reference height by using altitude and altitude-velocity measurements. The output of the controller is a throttle value that is sent to the motors resulting in an upward thrust.

Model for altitude control

A simplified version of Equation 3.2.21 is used and is presented in Equation 5.2.1.

$$\dot{w} = -g \cos\theta \cos\phi + F_{thrust} \tag{5.2.1}$$

In earth frame this corresponds to

$$\ddot{z} = -g + \cos\theta \cos\phi F_{thrust} \tag{5.2.2}$$

This is a double integrator with an offset caused by gravity. By rearranging Equation 5.2.2 the needed thrust is found as

$$F_{thrust} = \frac{\ddot{z} + g}{\cos\theta \cos\phi} \tag{5.2.3}$$

Since the direction of the thrust vector will not coincide with the earth frame z-axis if the roll or pitch angle differ from zero, division by $\cos\theta \cos\phi$ is made and thus an angle boost is applied to maintain the needed thrust.

Altitude controller

A cascaded Position → Velocity control structure is implemented according to Fig. 5.2.1, where the outer loop uses a PID controller that outputs a desired velocity based on the altitude reference, altitude measurements and velocity measurement,

$$\begin{aligned} u_z(t) &= K_p e_z(t) + K_i \textstyle\int_0^t e_z(\tau) d\tau - K_D \frac{d}{dt} z(t) \\ e_z(t) &= z_{ref}(t) - z(t) \end{aligned} \tag{5.2.4}$$

This desired velocity is used by the inner loop where a PI-controller calculates the needed change in throttle,

$$\begin{aligned} u_{\dot{z}}(t) &= K_p e_{\dot{z}}(t) + K_i \int_0^t e_{\dot{z}}(\tau) d\tau \\ e_{\dot{z}}(t) &= \dot{z}_{ref}(t) - \dot{z}(t) = u_z(t) - \dot{z}(t) \end{aligned} \tag{5.2.5}$$

The throttle offset needed to counter gravity depends on the battery level and this measurement resets every time the ArduCopter autopilot power is turned off, making it tricky to estimate. The current solution is that the user sets an approximate base throttle and then a rather large integral action is used to find the correct throttle needed for hover. On the first flight on a fresh battery the throttle offset is put to a rather low value and then the steady state throttle can be used as an initial value for the next flight. As an alternative, the user can set a throttle with the radio controller and then when the hexacopter goes into autonomous mode this throttle is sampled and used as the throttle offset.

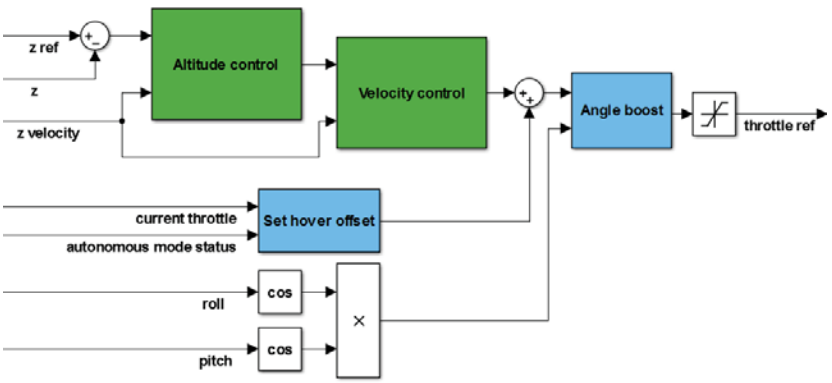

Figure 5.2.1.: Altitude controller

5.2.2. Horizontal control

The horizontal controller uses the roll and pitch angles to change the horizontal position and horizontal velocity.

Model for horizontal control

Again a simplified version of Equation 3.2.21 is used and since the position reference will be in earth frame the rotation matrix is used to express the linear accelerations

in earth frame. The included forces will be thrust and gravity but in earth frame gravity will only affect vertical acceleration along the z-axis and is thus not visible here, see Equation 5.2.6. Note that since only x and y are considered, only the first to rows of the rotation matrix is used,

$$\begin{bmatrix} \ddot{x}^E \\ \ddot{y}^E \end{bmatrix} = \mathbf{R}_B^E \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \mathbf{R}_B^E \begin{bmatrix} g\sin\theta \\ -g\cos\theta\sin\phi \\ -g\cos\theta\cos\phi + \frac{1}{m}F_{thrust} \end{bmatrix} \tag{5.2.6}$$

$$= \begin{bmatrix} (\cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi)\frac{F_{thrust}}{m} \\ (\sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi)\frac{F_{thrust}}{m} \end{bmatrix} \tag{5.2.7}$$

To simplify the equations a small angle approximation is used together with the approximation of the hexacopter operating near a thrust needed for hovering resulting in

$$\begin{bmatrix} \ddot{x}^E \\ \ddot{y}^E \end{bmatrix} \approx \begin{bmatrix} \theta\frac{F_{thrust}}{m} \\ -\phi\frac{F_{thrust}}{m} \end{bmatrix} \approx \begin{bmatrix} \theta\frac{mg}{m} \\ -\phi\frac{mg}{m} \end{bmatrix} = \begin{bmatrix} g\theta \\ -g\phi \end{bmatrix} \tag{5.2.8}$$

By inverting 5.2.8 the roll and pitch angles are expressed as a function of desired linear acceleration,

$$\begin{bmatrix} \phi \\ \theta \end{bmatrix} = \begin{bmatrix} -\frac{1}{g}\ddot{y} \\ \frac{1}{g}\ddot{x} \end{bmatrix} \tag{5.2.9}$$

5.2.2.1. Horizontal controller

Again a cascaded Position → Velocity controller is used according to Fig. 5.2.2. This time the outer position loop uses a PID controller that outputs a desired velocity based on the current position error and velocity measurement, see Equation 5.2.10.

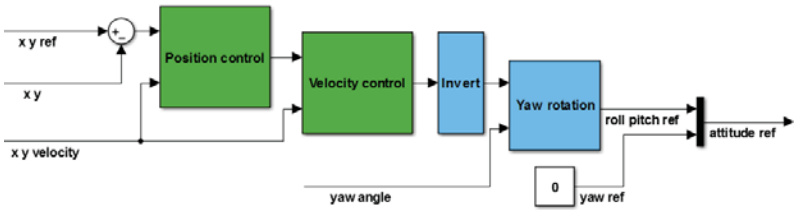

Figure 5.2.2.: Horizontal controller

$$\begin{bmatrix} u_x(t) \\ u_y(t) \end{bmatrix} = K_p e_z(t) + K_i \int_0^t e_z(\tau)d\tau - K_D \frac{d}{dt} y_{xy}(t)$$

$$e_z(t) = \begin{bmatrix} x_{ref}(t) - x(t) \\ y_{ref}(t) - y(t) \end{bmatrix}, \; y_{xy}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} \tag{5.2.10}$$

The inner velocity loop uses a PD-controller to calculate a desired roll and pitch angle from the velocity error,

$$\begin{aligned} \begin{bmatrix} u_{\dot{x}} \\ u_{\dot{y}} \end{bmatrix} &= K_p e_{\dot{z}}(t) + K_D \tfrac{d}{dt} e_{\dot{z}}(t) \\ e_{\dot{z}}(t) = \begin{bmatrix} \dot{x}_{ref}(t) - \dot{x}(t) \\ \dot{y}_{ref}(t) - \dot{y}(t) \end{bmatrix} &= \begin{bmatrix} u_x(t) - \dot{x}(t) \\ u_y(t) - \dot{y}(t) \end{bmatrix} \end{aligned} \tag{5.2.11}$$

The coupling between roll and pitch angle and x- and y-acceleration was presented in Equation 5.2.9 and is the reason for the "invert" block in Fig. 5.2.2. The position and velocity control is made in earth frame so the roll and pitch setpoints have to be rotated by the yaw rotation matrix because the ArduCopter autopilot operates in body frame. It is possible to reach any position without changing the yaw angle and since no sensor (camera etc.) that needs a specific yaw angle is used, its setpoint will always be set to zero.

6. Simulink model

Simulink [28] is an extension to Matlab that provides a block diagram environment used for simulation and model-based design. With the help of different tool-boxes, the user is provided with a number of blocks that can do anything from a multiplication to advanced features like computer vision or nonlinear control. Own blocks can be created and Matlab code can also be included. It is an hierarchical environment where blocks can be included within other blocks. A support package is available for the PandaBoard [29]. This makes it possible to run a Simulink model on the board and also enables the use of special blocks for interacting with other hardware. The PandaBoard does not run a Matlab Simulink version itself but instead the model is turned into C-code by automatic code generation.

6.1. Main blocks

Here the blocks on the highest hierarchical level are presented. An overview can be seen in Fig. 6.1.1.

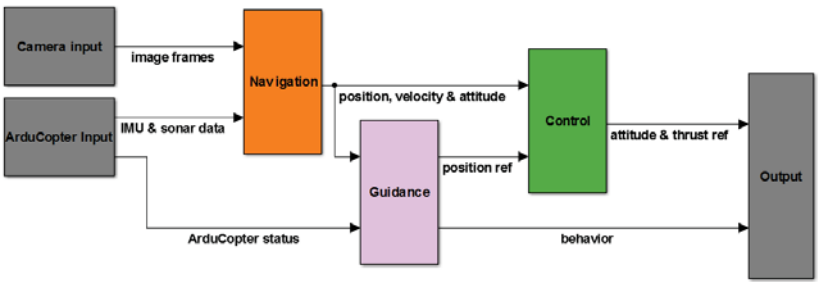

Figure 6.1.1.: Simulink model overview

- Input: Decodes the messages from the ArduCopter autopilot and also performs some appropriate scaling and removal of initial signal bias.
- Camera input: Receives camera frames.
- Navigation: Contains the computer vision parts and the unscented Kalman filter.

- Guidance: User setpoints are sent to the controllers and the block also has a "brain" that was developed in [1] that can make decisions and set the desired behavior of the hexacopter.
- Control: Contains the altitude and horizontal controllers.
- Output: Sends the attitude setpoints, throttle setpoints and current status to the ArduCopter autopilot.

6.2. Scheduling

Different parts of the system have different time constraints and their computational demand varies. To handle this, the "function-call" block is used which triggers selected blocks at a user defined rate. Three different rates are used:

40Hz This is the fastest rate and is used by the input block, the unscented Kalman filter and the controllers.

20Hz This is the medium rate which is used by the vision blocks and the guidance blocks.

10Hz The slowest rate, used for plotting.

The system only runs in soft real time so there is no guarantee that the desired rates are held.

7. Experiments

To validate the system a number of experiments are carried out both in simulation and on the real process.

7.1. Simulation

Before performing experiments on the real system, evaluation and tuning of the estimation and control algorithms were made by simulation. The Simulink model presented in Chapter 6 is used but the inputs and outputs now are now connected to a virtual model instead of the real system. This includes:

- Hexacopter model: Here the linear accelerations are calculated as described in Equation 3.2.21 and the attitude is determined by the system identification described in Sec. 3.3.1. To simulate battery drop, the generated thrust for a certain throttle is lowered by time.
- Sensor model: Noise and biases are added to the measured signals to simulate the sensor outputs of the real system. The noise is band-limited white noise with covariances chosen according to Sec. 4.2.3. Rather high noise levels are added to the roll, pitch and yaw to simulate the shaking and vibration of the hexacopter while flying, this is not only applied to measurements but also affects the translational acceleration.
- Camera simulator: In [1] a camera simulator was developed that outputs image frames based on position and attitude. It also adds pixel noise based on velocity.

Altitude estimation and control are evaluated by looking at a step change.

To illustrate the benefits of sensor fusion, horizontal position estimation and control are evaluated in simulation by performing a series of steps in horizontal position where the step signals are used as a reference for the position controller. Note that step transitions are made in two dimensions simultaneously, one along the x-axis and one along the y-axis. For each experiment the SRUKF is used for estimation but the available measurements are changed. First the computer vision input to the filter is disabled so that the filter only can use the estimated attitude and accelerometer readings to estimate horizontal position. In the second experiment the filter ignores the attitude and accelerometers and instead relies entirely on the estimated position from the computer vision algorithm. In the third and final experiment all available

information is used by the filter so that computer vision data is fused with attitude estimates and accelerometer readings.

7.2. The real system

When a satisfying performance was achieved in simulation, experiments on the real system were carried out. The hexacopter was switched into autonomous mode standing on the ground with the goal of trying to hold its current horizontal position at a reference altitude for a minute. Autonomous take-off and landing were also included in the tasks.

A comparison was made offline between the SRUKF and the EKF by feeding both filters with the same logged data, using the same model and the same covariance matrices.

8. Results

This chapter presents the results of the experiments carried out in Chapter 7.

8.1. Simulation

The results of the estimation and control from simulation are presented in the following subsections.

8.1.1. Estimation

The altitude estimation is presented in Fig. 8.1.1 and has a very low deviation from the truth value thanks to the sonar.

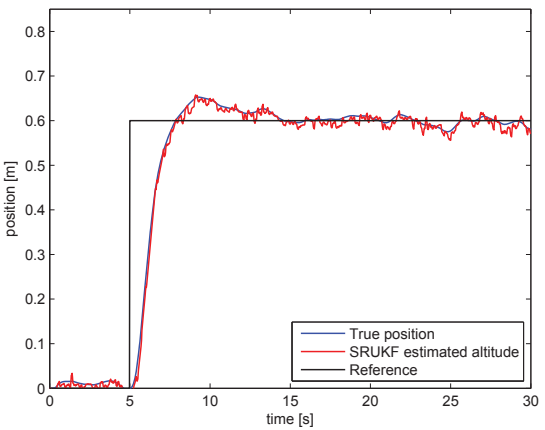

Figure 8.1.1.: Altitude estimation

The result of the first horizontal step change experiment is shown in Fig. 8.1.2. The position controller makes the estimated position follow the reference but the true position drifts away because of high drift in the position estimation when integrating

velocity estimates based on noisy measurements. Note that there are four curves representing the position estimation in the plot because step changes are made both along the x-axis and the y-axis simultaneously.

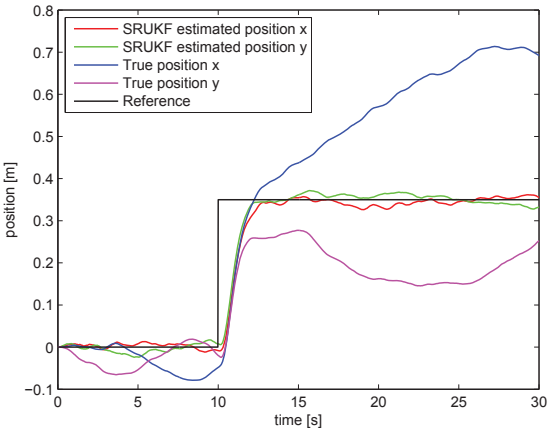

Figure 8.1.2.: Position estimated based only on IMU data

The second horizontal step change experiment can be seen in Fig. 8.1.3. Again only the estimated position follows the reference because of drift in estimation, this time the drift is caused by short blackouts in the vision estimation when a new template has to be chosen.

The third horizontal step change experiment is presented in Fig. 8.1.4. This time the drift the has been reduced significantly as the vision algorithm is used but the short blackouts are taken care of by integrating the estimated velocity.

The estimated velocities are shown in Fig. 8.1.5 and Fig. 8.1.6.

8.1.2. Control

Altitude and horizontal control are evaluated using the plots already presented in Sec. 8.1.1.

Fig. 8.1.1 shows that the altitude is subject to a small overshoot at the step change but that it is kept within 3 cm from the reference in steady state conditions. The integrator is still large enough to counter the simulated battery voltage drop.

The horizontal control is stable and keeps the estimated position in Fig. 8.1.2 to Fig. 8.1.4 within 5 cm from the reference with a very small overshoot at the step changes.

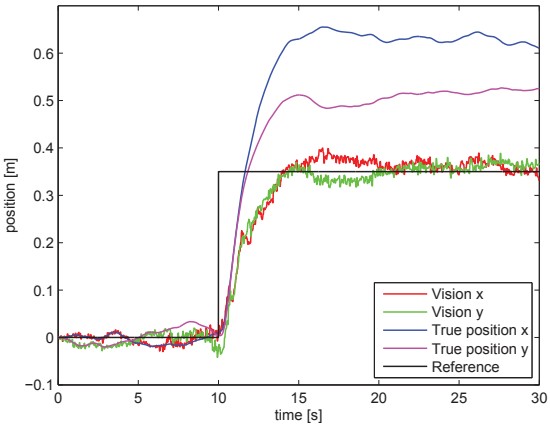

Figure 8.1.3.: Position estimation based only on vision data

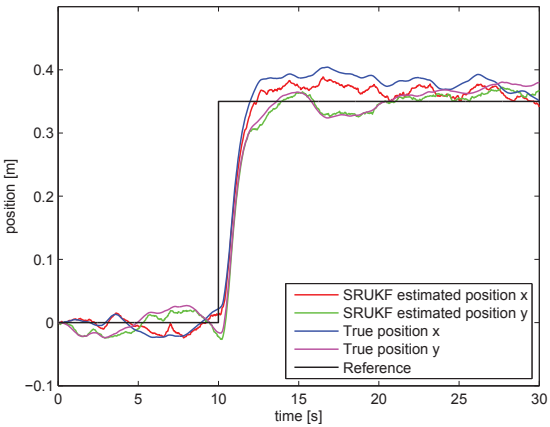

Figure 8.1.4.: Position estimation based on sensor fusion of IMU and vision data

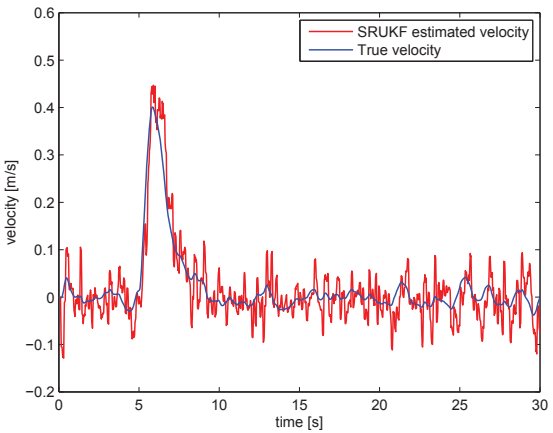

Figure 8.1.5.: Altitude velocity estimation

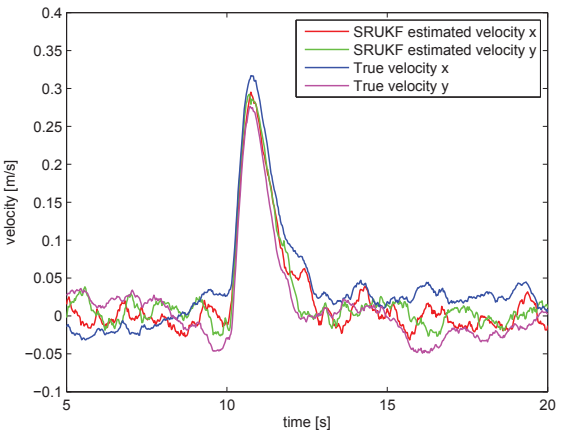

Figure 8.1.6.: Horizontal velocity estimation

8.2. The real system

The results of the estimation and control on the real system is presented in the following subsections, since no truth data is available some of the estimates are compared to sensor measurements and vision data to prove the difference. The hexacopter can now hold its altitude and horizontal position better then before the work of this thesis and is able to hover in place for a minute without problem most of the time. The main issues are lags and loss of communication between the PandaBoard and the ArduCopter autopilot, which might have to do with the PandaBoard not being able to run all its tasks fast enough. The system still depends a lot on the vision and is sensitive to fast movement, it can recover but will loose track of its position.

8.2.1. Estimation

The estimated altitude is presented in Fig. 8.2.1. The sonar measurement is given a high trust in the SRUKF so the estimated altitude is a smoothed version of the sonar.

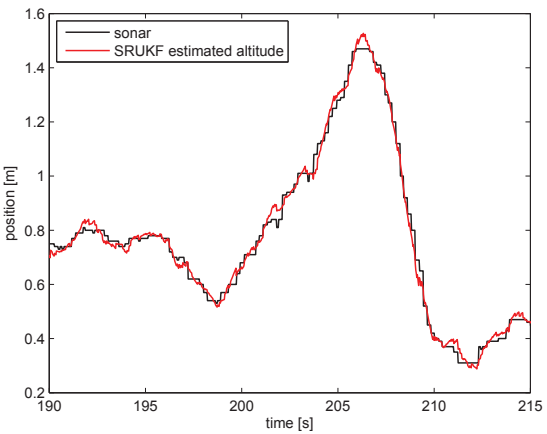

Figure 8.2.1.: Altitude position estimate, comparison between SRUKF estimate and sonar measurement

The altitude velocity estimate is compared to the derivative of the sonar measurement and is shown in Fig. 8.2.2. The SRUKF estimate is much smoother but still without the delay and dampening that is introduced by using a simple low-pass filter.

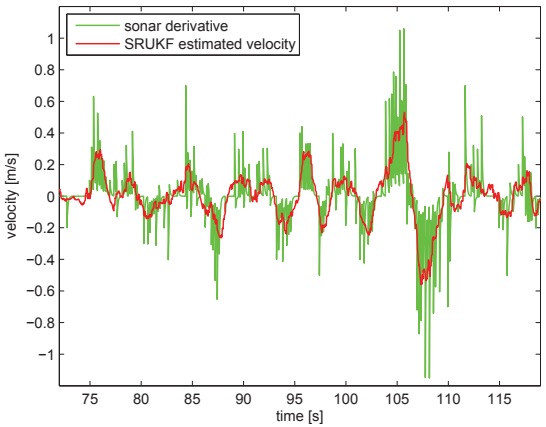

Figure 8.2.2.: Altitude velocity estimate, comparison between SRUKF estimate and sonar measurement derivative

For horizontal velocity estimation, quite a lot of trust is given to the model and therefore it is sensitive to the roll and pitch angles. At low velocities the SRUKF velocity estimates are a smoothed version of the derivative of the vision estimates, see Fig. 8.2.3 and Fig. 8.2.4 but at high velocities the vision derivatives reacts very little and the difference is shown in Fig. 8.2.5.

The horizontal position estimation is hard to evaluate but at very low velocities it is rather stable. As soon as the speed goes up so does the drift, this makes trajectory following impossible at least in such a tight space as the test location.

As can be seen in Fig. 8.2.6 the filter outputs of the EKF and the SRUKF are almost identical but the EKF showed to be far from as robust as the SRUKF and would sometimes diverge.

8.2.2. Control

The altitude controller is able hold the altitude within 5-10 cm from the reference most of the time but air turbulence causes deviations and it is also problematic to find the throttle needed for hover since it depends a lot on the battery level. A plot of the altitude controller performance is shown in Fig. 8.2.7 where it takes off automatically and tries to hold the altitude at a 65 cm reference. Notice that the altitude starts at 0.2 m because this is the minimum range of the sonar.

The horizontal position controller performance is presented in Fig. 8.2.8. The position usually stays within 15 cm from the reference. It should be noted that this is

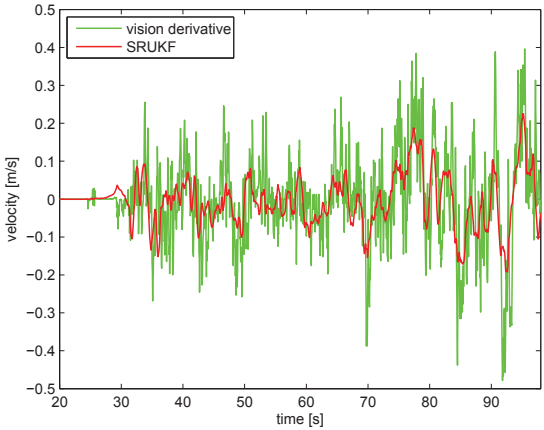

Figure 8.2.3.: Velocity x estimate, comparison between SRUKF estimate and derivative of vision estimate

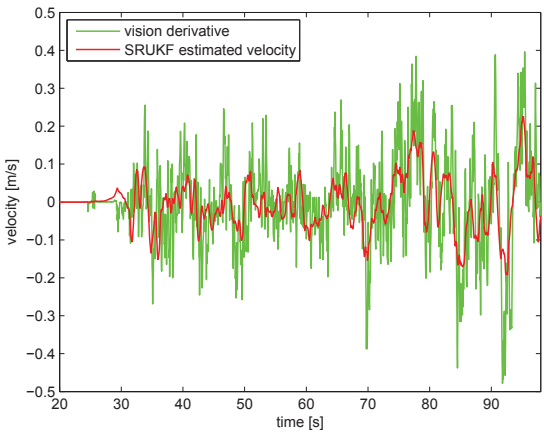

Figure 8.2.4.: Velocity y estimate, comparison between SRUKF estimate and derivative of vision estimate

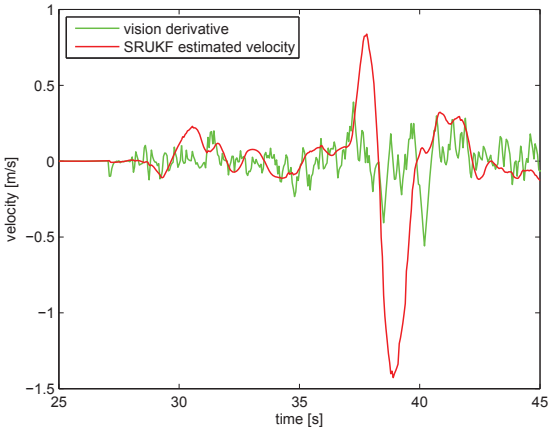

Figure 8.2.5.: Velocity x estimate, comparison between SRUKF estimate and derivative of vision estimate when performing a fast maneuver

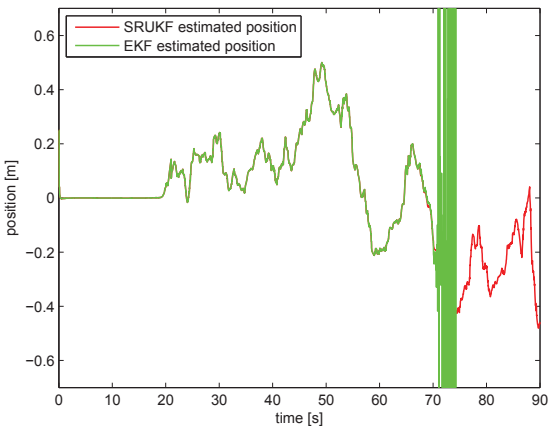

Figure 8.2.6.: Comparison between SRUKF and EKF, estimation of horizontal position x

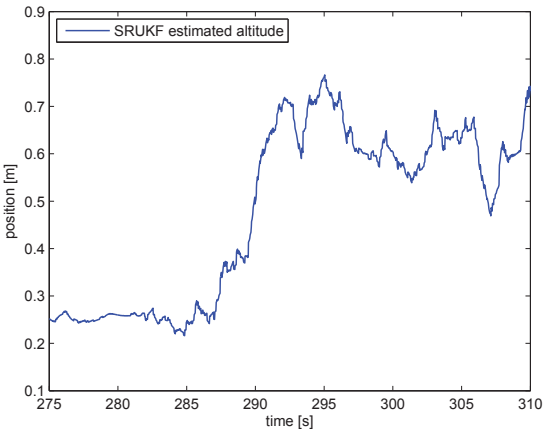

Figure 8.2.7.: Altitude control with a setpoint of 0.6m

estimated position and not true position so if the position estimate drifts it will not show, but that is a question of estimation and not control.

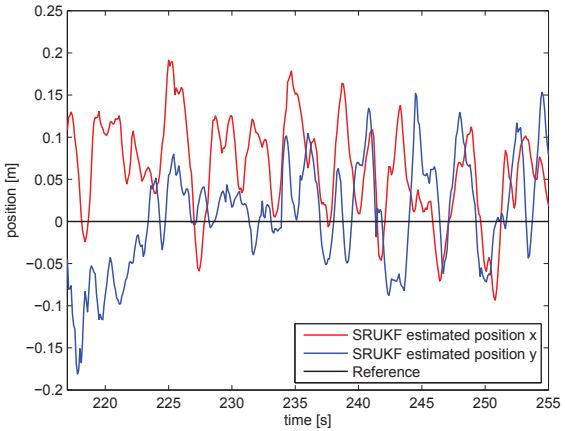

Figure 8.2.8.: Horizontal position control

9. Discussion

9.1. General

The hexacopter is able to take off and hover autonomously at a fixed position with some deviations caused by air turbulence and errors in the position and velocity estimation. Flights are more stable and position is held better now compared to before the work that was carried out in this thesis due to improvements in both estimation and control. Still the estimation is only able to keep the hexacopter in a hovering state and not follow trajectories. It should be noted that all test flights are carried out in a very tight space and the hexacopter is only allowed to move within an area of 1.3x2m so if one would fly over a larger area at a higher altitude the result might be better.

9.2. Estimation

The SRUKF provides much smoother velocity estimates compared to differentiation of the raw sensor measurements without infusing the large delay that can be a problem when using a simple low-pass filter, resulting in an increased controller performance. For the horizontal position estimation the absence of a more absolute measurement is still an issue and a major problem that was discovered in late testing is that the image frames from the camera are affected by a varying time delay of about 0.1-1 seconds. The delay was approximated by looking at logged data and comparing the attitude measurement to the camera image for a fast movement. It is unclear if the time delay is created at the web camera itself or at the input to the PandaBoard. The vision algorithm has to pick a new template very often because no good match can be found and even if the velocity is integrated to fill the gaps for the time when the new template is chosen, estimation is still very sensitive to fast movement.

The EKF would some times diverge and the main problem seems to be the vision measurements, perhaps some smoothing of the vision measurements are required to prevent problems with the calculation of the Jacobian. The SRUKF never diverged during simulation, real flights or when applied to offline data. The downside with the SRUKF implementation is that the required computation time is almost five times the computation time of the EKF with the high computational complexity being related to the update of the covariance matrices.

9.3. Control

The altitude controller shows a more stable performance then the previous ArduCopter autopilot controller, mainly because of improved velocity information from the SRUKF. Since all testing is done in a very tight space air turbulence causes deviations and battery-level dependency makes it difficult to find the correct throttle needed for hover without using a large integral part in the controller. The large integral part can induce some fluctuations in the altitude in combination with the air turbulence but never leads to instability. The alternative would be to map battery level to the hover throttle but the only battery measurement available resets every time the power is turned off and to be able to use different batteries without changing parameters it is more convenient to just have a larger integral part.

The horizontal controller keeps the estimated position within an acceptable range in a stable manner. The derivative part in the velocity controller showed to be a very important parameter for stability.

10. Conclusion

Improvements have been made without using any additional hardware or modifying the vision algorithm. The use of SRUKF mainly improved velocity estimates but also position estimates, both contributing to flights being more stable and with less drift compared to only using vision data for position estimation. The square root implementation of the SRUKF is very robust and has never diverged or showed any strange behavior in either simulation or real test for any combination of parameters. The EKF on the other hand was not as robust and had some problems with the vision measurements which sometimes led to divergence when applied to offline data. When divergence was not a problem, the difference between the EKF output and the SRUKF output was negligible. With the current implementation the SRUKF suffers from a high computational complexity compared to the EKF.

The cascaded control structures presented have proved to be working well, ensuring stability of the closed loop system. Increasing the throttle correctly over time to compensate for battery loss was one of the most challenging parts since the battery discharge is nonlinear and its effect on the throttle is very large compared to the amount of throttle needed for stabilization.

The filter was not able to compensate for the weaknesses in the position estimation of the vision system when the hexacopter is moving fast and with no absolute horizontal position measurement drift will always be an issue. As a result the hexacopter can only hover in a fixed position and tasks like following a designated path is not possible with the current position estimation, at least not when the space is so narrow. The system only runs in soft real time and seems to have problems with processing all the needed tasks, causing lags and communication problems. Large, varying time delays in the image frames also degrades both estimation and control. Optimization of the Simulink model and implementation of a hard real-time system should be seen as future work.

A. The extended Kalman filter algorithm

The algorithm for the extended Kalman filter is presented below [23].

- Initialize, $k = 0$

$$\hat{\mathbf{x}}_0 = E[\mathbf{x}_0],\ \mathbf{P}_{x_0} = chol\{E[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^T]\} \tag{A.0.1}$$

$$\mathbf{Q} = E[(\mathbf{v} - \bar{\mathbf{v}})(\mathbf{v} - \bar{\mathbf{v}}))^T],\ \mathbf{R} = E[(\mathbf{n} - \bar{\mathbf{n}})(\mathbf{n} - \bar{\mathbf{n}}))^T] \tag{A.0.2}$$

- For $k = 1, ..., \infty$

  1. Calculate Jacobians

     The filter requires linear process and measurement models when predicting and updating the covariance matrices and Kalman gain $\mathbf{K}_k$. This is done by using a first order Taylor expansion.

$$\mathbf{F}_{k-1} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}} \tag{A.0.3}$$

$$\mathbf{H}_k = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k|k-1}} \tag{A.0.4}$$

  2. Predict

The nonlinear process model is used to predict the state.

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}) \tag{A.0.5}$$

And the linearized process model is used to predict the covariance of the estimate.

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1} \tag{A.0.6}$$

3. Update

The predicted state is propagated through the nonlinear measurement model and this predicted measurement is then compared to the real measurement, forming a measurement residual.

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{h}(\hat{\mathbf{x}}_{k|k-1}) \tag{A.0.7}$$

The covariance of the residual is calculated by using the linearized measurement model and the predicted state covariance

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k \tag{A.0.8}$$

The Kalman gain is formed. If no information had been lost by the linearization, a perfect model is assumed and disturbances are white noise, this would had been optimal

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1} \tag{A.0.9}$$

The state is updated by using the predicted state and the measurement weighted by the Kalman gain.

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k \tag{A.0.10}$$

The covariance is updated

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{A.0.11}$$

Bibliography

[1] N. Ohlsson and M. Stahl, "A Model-Based Approach to Computer Vision and Automatic Control using Matlab Simulink for an Autonomous Indoor Multirotor UAV," 2013.

[2] Wikipedia: Unmanned Aerial Vehicle. [Online]. Available: http://en.wikipedia.org/wiki/UAV

[3] DIY Drones. [Online]. Available: http://diydrones.com/

[4] Aeroquad: The open source quadcopter / multicopter. [Online]. Available: http://aeroquad.com/

[5] S. Lange, N. Sunderhauf, and P. Protzel, "A vision based onboard approach for landing and position control of an autonomous multirotor UAV in GPS-denied environments," in *Advanced Robotics, 2009. ICAR 2009. International Conference on, year = 2009, pages = 1–6, organization = IEEE.*

[6] I. Sa and P. Corke, "System identification, estimation and control for a cost effective open-source quadcopter," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on, year = 2012, pages = 2202–2209, organization = IEEE,.*

[7] R. W. Beard, "Quadrotor dynamics and control," *Brigham Young University*, 2008.

[8] R. C. Leishman, J. Macdonald, R. W. Beard, and T. W. McLain, "Quadrotors & Accelerometers," 2013.

[9] J. Macdonald, R. Leishman, R. Beard, and T. McLain, "Analysis of an Improved IMU-Based Observer for Multirotor Helicopters," *Journal of Intelligent & Robotic Systems*, pp. 1–13, 2013.

[10] S. J. Julier and J. K. Uhlmann, "A general method for approximating nonlinear transformations of probability distributions," *Robotics Research Group, Department of Engineering Science, University of Oxford, Oxford, OC1 3PJ United Kingdom, Tech. Rep*, 1996.

[11] R. Van Der Merwe, "Sigma-point Kalman filters for probabilistic inference in dynamic state-space models," Ph.D. dissertation, University of Stellenbosch, 2004.

[12] S. M. Siddiqui, "Integrated navigation and self alignment using Square Root Unscented Kalman filtering," in *Applied Sciences and Technology (IBCAST),*

2013 10th International Bhurban Conference on, year = 2013, pages = 73–76, organization = IEEE,.

[13] R. Kandepu, B. Foss, and L. Imsland, "Applying the unscented Kalman filter for nonlinear state estimation," *Journal of Process Control*, vol. 18, no. 7, pp. 753–768, 2008.

[14] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, "PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision," *Autonomous Robots*, vol. 33, no. 1-2, pp. 21–39, 2012.

[15] DIY Drones: APM:Copter. [Online]. Available: http://copter.ardupilot.com/

[16] S. Bouabdallah, "Design and control of quadrotors with application to autonomous flying," *Ecole Polytechnique Federale de Lausanne*, 2007.

[17] N. Guenard, T. Hamel, and R. Mahony, "A practical visual servo control for an unmanned aerial vehicle," *Robotics, IEEE Transactions on*, vol. 24, no. 2, pp. 331–340, 2008.

[18] 3DR Robotics Website. [Online]. Available: http://3drobotics.com/

[19] PandaBoard ES Website. [Online]. Available: http://pandaboard.org/content/pandaboard-es

[20] R. M. Murray and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[21] P. Martin and E. Salaun, "The true role of accelerometer feedback in quadrotor control," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on, year = 2010, pages = 1623–1629, organization = IEEE, owner = fogelberg,.*

[22] R. E. Kalman *et al.*, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[23] S. Särkkä *et al.*, *Recursive Bayesian inference on stochastic differential equations*. Helsinki University of Technology, 2006.

[24] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[25] R. Leishman, J. Macdonald, T. McLain, and R. Beard, "Relative navigation and control of a hexacopter," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on, year = 2012, pages = 4937–4942, organization = IEEE,.*

[26] S. Saripalli, J. M. Roberts, P. I. Corke, G. Buskey, and G. S. Sukhatme, "A tale of two helicopters," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 1. IEEE, 2003, pp. 805–810.

[27] K.-E. Årzén, *Real-time Control Systems*. Department of Automatic Control, Lund Institute of Technology [Institutionen för reglerteknik, Tekniska högsk.], 2011.

[28] SIMULINK, Simulation and Model-Based Design. [Online]. Available: http://www.mathworks.se/products/simulink/

[29] MathWorks: PandaBoard Support from Simulink. [Online]. Available: http://www.mathworks.se/hardware-support/pandaboard.html

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
December 2013

Document Number
ISRN LUTFD2/TFRT--5930--SE

Author(s)
Johan Fogelberg

Supervisor
Simon Yngve, Combine Control Systems
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Karl-Erik Årzén, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle

Navigation and Autonomous Control of a Hexacopter in Indoor Environments

Abstract

This thesis presents methods for estimation and autonomous control of a hexacopter which is an unmanned aerial vehicle with six rotors. The hexacopter used is a ArduCopter 3DR Hexa B and the work follows a model-based approach using Matlab Simulink, running the model on a PandaBoard ES after automatic code generation. The main challenge will be to investigate how data from an Internal Measurement Unit can be used to aid an already implemented computer vision algorithm in a GPS-denied environment.

First a physical representation is created by Newton-Euler formalism to be used as a base when developing algorithms for estimation and control. To estimate the position and velocity of the hexacopter, an unscented Kalman filter is implemented for sensor fusion. Sensor fusion is the combining of data from different sensors to receive better results than if the sensors would have been used individually. Control strategies for vertical and horizontal movement are developed using cascaded PID control. These high level controllers feed the ArduCopter with setpoints for low level control of angular orientation and throttle.

To test the algorithms in a safe way a simulation model is used where the real system is replaced by blocks containing a mix of differential equations and transfer functions from system identification. When a satisfying behavior in simulation is achieved, tests on the real system are performed.

The result of the improvements made on estimation and control is a more stable flight performance with less drift in both simulation and on the real system. The hexacopter can now hold position for over a minute with low drift. Air turbulence, sensor and computer vision imperfections as well as the absence of a hard realtime system degrades the position estimation and causes drift if movement speed is anything but very slow.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-75

Recipient's notes

Security classification

http://www.control.lth.se/publications/