NTNU – Trondheim
Norwegian University of
Science and Technology

Visual Navigation of an autonomous drone

Pål Moen Møst

Master of Science in Informatics
Submission date: February 2014
Supervisor: Helge Langseth, IDI
Co-supervisor: Boye Annfelt Høverstad, IDI
Christian Schellewald, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Pål Moen Møst

Visual Navigation of an autonomous drone

Specialisation project, Spring / fall 2013

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

NTNU
Det skapende universitet

Abstract

In this thesis we have added and developed the necessary hardware and software to transform a radio controlled quadcopter into an autonomous drone.

The main goal for this thesis was to precise and safely land the drone on its lanidng spot. This should be done autonomously and by visual navigation. To make this possible the we started with the assembly of a radio controlled quadcopter from a almost-ready-to-fly kit (ARF) based ont he AeroQuad platform.

We then applied the necessary hardware to the drone by adding a web camera for visual abilities, an embedded computer for processing power and wireless access point for telemetry.

We implemented computer vision algorithms the enable visual navigation and the linear Kalman filter to give robustness to the visual tracking and to cope with noisy sensor observation. A proportional-integral-derivative controller (PID controller) was developed to control the drones pose and fly to desired positions. These implementations was tested in experiments in a real-world environment and the drones ability to perform autonomous tasks based on visual navigation are presented in results.

Preface

This masters thesis was written by Pål Moen Møst during 2013 and January of 2014. The work is the final report of the Master of Science in Artificial intelligence and learning at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU). I would like to thank my supervisor Helge Langseth and co supervisors Boye Annfelt Høverstad and Christian Schellewald for their guidance and support. I also like to thank the technical Staff at IDI for their support regarding equipment and hardware.

Pål Moen Møst
Trondheim, February 1, 2014

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last decade drones have moved from mostly being used in military operations to being demilitarized and available for the public market. We can see to the rise of the smart phone to explain this drone revolution that have been going the last couple of years and is still going on. The smartphones demand for more processing power and low power consumption have resulted in new hyper efficient single-chip-processors based on the RISC architectures led by the chip designer ARM. These processors as it turns outs are also well suited for use as autopilots in drone and the huge demand for smartphones have rapidly driven the prices of single-chip processors down. The smartphone-drone connection goes far beyond only the processors, the smartphones today are packed with hardware that we can also need in drone autopilots. Tnternal sensors like accelerometers, gyroscopes and GPS can be found in almost every smartphone today, a swell as high quality image sensors. The smartphone market have driven the prices and size for this hardware at an all time low. With these cheap and powerful hardware available to public the drone revolution have literally lifted off and drones have been used in a wide variety of applications. In agriculture for monitoring farm crop, in rescue operations, surveillance of power grids and delivery systems. The drones used in real-world applications are mostly radio controlled by humans with the support of an autopilot. As the hardware and software matures we will probably see more fully autonomous drones in real-world applications in the future.

1.1 Background and Motivation

In the last five years there have been a great deal of drone projects and some with mind blowing results. Everything from aggressive formation flying micro drones to pole-throwing and catching drones. These projects have been conducted with

high-end equipment in the range of hundreds thousand of dollars. This thesis will focus on the possibility of designing a autonomous drone using only off-the-self available low-cost hardware and open source software. The drone will be acting in a real-world environment and should be able to preform real-world tasks like the goals described in Section 1.2.

1.2 Goals and Research Questions

Research question 1 *How well will a drone designed with off-the-self hardware, open source software perform in autonomous flight with navigation with computations done on on-board hardware.*1

Research question 2 *What AI-methods are necessary implemented to make this drone autonomous?*

Goal *Autonomously land the drone on a given landing spot*

Sub goals

- Build the drone and make it operational, airborne.
- Control the drone with code running high level on-board hardware.
- Preform stable altitude hold.
- Preform stable position hold over the marker.
- Land safe and precisely on the marker.

1.3 Research Method

The research method for this this will be design/experiment. The reason for this is the uncertainty of how well the hardware and software will work independently and more import, together.

Chapter 2

Background Theory and Motivation

2.1 Motivation

2.1.1 Build our own drone

At the commercial market today there are not many options to buy a drone that fits the criteria for this thesis. The criteria are that the drone should be able to operate independently without any need for a ground station or radio control. The drone should have camera abilities and be able the run computer vision on-board. The drones that are commercial available at the market today and able to perform this kind of tasks are in a very high price, from $6000 and above. This requires a big budget, but even if money where no issue most of these expensive drones are deployed with proprietary software software running on-board.

2.1.2 Autonomous precision landing

The idea of autonomous precision landing was based on issues the electric power company Trønder Energi had with landing their radio controlled drones used for surveillance of power grids should land on a trailer. The motivation choosing this goal was after some research found that this is a missing feature in most commercial available drones today within the price range of $1000. Limited battery power is one of the main limitations of drones today, usual flight time of these kind drones today is about 10 to 25 minutes depending on their current pay load. This feature possibly enable drones to land on charging station and make them completely autonomous. Autonomous drone with this feature can possibly

be used for motoring power grids in distance locations with nearby charging station.

2.2 Background Theory

2.2.1 Control theory

To control a dynamic system a system that changes state over time control theory is applied. Control theory is about implementing a controller that helps the system reach a desired state. The controller calculates the corrective action to help the system reach this state also known as a set point. The corrective actions $u(t)$ are calculated based on the error $e(t)$. The error is measured with the difference between the system state $x(t)$ and set point $s(t)$. This goes into a generic feedback loop that continuously measures the error and calculates the corrective action and trying to hold the system in a stable state at the given set point without any oscillations. This is schematically represented Figure **??**.

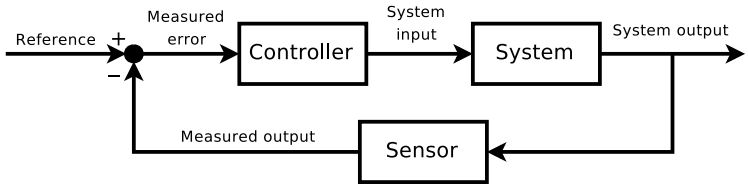

Figure 2.1: Basic feedback loop for control of a dynamic system. This figure is taken from [2] and licensed under Creative Commons Attribution 2.5 license

2.2.2 PID Controller

The PID controller is a widely used and well known controller used by the industry today. PID is an acronym for the terms the controller uses, Proportional, integral and derivative. It is based on the same idea as the generic feedback controller described in the previous section, but with extensions. If you ever have used cruise control on a car there is a high possibility that it is controlled by a PID controller. Where the set point is the your wanted speed. A car will be subjected to a great deal of external disturbances caused by the environment it is acting in. This could cause oscillations around the set point. If a car using cruise control is driving up a step hill the car will need more throttle to reach the set point and

[2] `http://en.wikipedia.org/wiki/File:Feedbacka_loop_with_descriptions.svg`

keep it in comparison to a car driving on a straight road. To cope with this kind of random disturbances the PID controller takes the present, the past and the future into consideration and are represent by the terms P , I and D.

- The proportional term is the current error $Kpe(t)$, where Kp is a weighting parameter
- The integral is the accumulated past error $Ki \int_0^t e(t)dt$. where Ki weighting parameter.
- The derivative term defines the slope of error over time $Kd\frac{d}{dt}e(t)$ and gives a prediction for the future error.

The corrective system input $u(t)$ is calculated by.

$$u(t) = Kpe(t) + Ki \int_0^t e(t)d + Kd\frac{d}{dt}e(t) \tag{2.1}$$

Depending on the complexity of the application where the PID controller is implemented different designs of the controller could be applied by adding or removing one or more of the three terms.

Proportional term

This term is mandatory in every controller, its main objective is to reduce the error. The higher the measured error is, the stronger the controller signal from the P term will be proportional to the gain Kp. The higher the gain the faster it will reach its set point, but to high of a gain will the cost of it overshooting and the time its takes to settle around the set point may be grater than a controller with lower gain. In more complex applications where there are great deal of external disturbances a purely P based controller will not work at all, causing the systems to overshoot and oscillate around the set point and never settle.

Integral term

This term is implemented to help the controller keep a steady state around the given set point. In the case that the P and PD controller settles below or above the given set point. The I term will help eliminate this bias using the accumulated error over time t . If the system for some reason suffers from saturation the I term will become very large and leads to unwanted behaviour. o handle this windup effect, constraints could be applied to the I term to prevent it from accumulating to large.

Derivative term

The derivative term is applied to dampen the oscillation caused by the P term. Its dependent on the rate of change in error, the higher the rate is , the more it will contribute to the $u(t)$ system input. This term could be very useful in difficult applications, but it is also the hardest to tune.

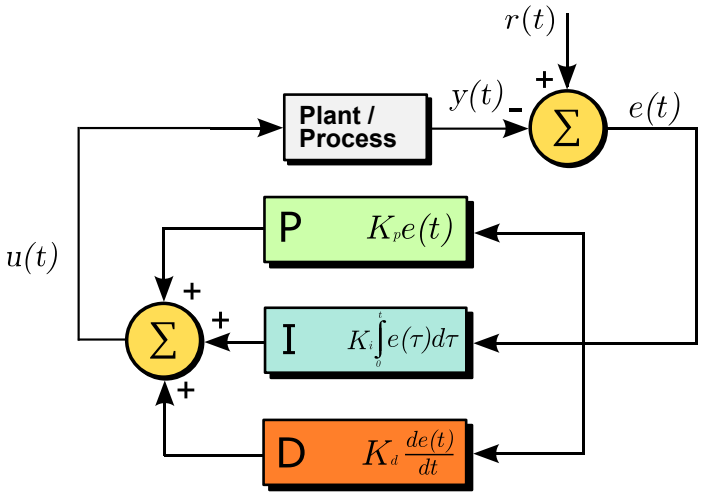

Figure 2.2: Block diagram on the PID feed back loop process This figure is taken from [4] and licensed under Creative Commons Attribution 2.5 license

Tuning parameters

For dynamic system to reach a stable state the PID controller needs to be tuned. The PID controller is only as good as its tuning parameters. The tuning parameters are.

- Kp is the gain for the proportional term
- Ki is the gain for the integral term.
- Kd is the gain for the derivative term.
- Windup guard constraint for the integral term.

[4] `http://commons.wikimedia.org/wiki/File:PID_en_updated_feedback.svg`

In most systems a Kp and Kd are sufficient gains to tune. The Kp gain is indicates how fast the state should reach its set point. In most cases we want the system to reach the state as fast as possible, but if we set the Kp gain to high, the cost could be that the controller is overshooting. To cope with overshooting, the Kp gain could be increased introduced to dampen the effect of the Kp when its closes in on the set point.

Steady state error

If the state settles above or bellow the set point and continues this stagnation. The error between the set point and the steady state is refereed to as an steady state error. To resolve this stagnation a the Ki parameter could be increased.

2.2.3 Sensor fusion

In real-world environments a robot is not able to directly observe the world state. Robots use sensors to give an perception of the world. The limitations with observing the world through sensor are that they are subjected to noise and the measurements therefore maybe inaccurate, sensors could also malfunction for short periods of time. No sensors is prefect, so to give a more accurate estimate of the world the robots often uses multiple sensors and fuses the observations together. This is what we call sensor fusion and the key idea is to use measurements from multiple sensors to correct and verify each other and combining the observations to give a better and more accurate estimate of the world. There is a vary of filters that can be applied for these kind of applications. A well known data fusion algorithm for real-world applications is the filter.

2.2.4 The Kalman filter

The Kalman filter is one of the most important and widely used data fusion algorithms in the industry today. It is was developed over fifty years ago by Rudolf E. Kalman [Kalman, 1960]. One of one of most famous and historic use of the Kalman filter was in the navigation systems of the Apollo 11 that put Neil Armstrong on the moon. Today Kalman filter is often used in GPS navigation system, drones and smartphones. The success of the Kalman filter may be its ability to model the observations from noisy observations and system dynamics in real-time and give a more accurate estimate of the wo1ld-state. In the devices containing sensors like GPS, Accelerometers and gyroscopes is often small embedded devices with limited computational power, Kalman filter is often preferred in this kind of solutions due to its small computational cost. The idea of the Kalman filter is to incorporate our predicted state with our estimation to give a more accurate estimate. The Kalman filter uses a set of previous observations

to give estimate of the current state. The Kalman filter assumes that observations are Gaussian distributed and that they are subject to white Gaussian noise and that the system is linear. The state is represented as a normal distribution. The state could be estimate with the linear stochastic difference Equation 2.2 the Equation 2.3 is the observation update and Equation 2.5 and 2.5 represents the noise for the process and observations as a normal distribution.

$$x_t = A_t x_{t-1} + Bu_t + w_t \tag{2.2}$$

$$Z_t = H_t x_t + v_t \tag{2.3}$$

with

$$w_t \ N(0, Q) \tag{2.4}$$

$$v_t \ N(0, R) \tag{2.5}$$

- x_t is the state vector it represents the state parameters of the system. Example of state parameters are; position, velocity, acceleration, heading at time t
- A_t is the state transition matrix maps the impact the state parameters in x_{t-1} applies to the state parameters in x_t. For example the system position and velocity will both influence the estimated position at time t.
- u_t is the vector that represents control input applied to the state at t.
- B_t is the control matrix maps the impact the control vector u_t has on the state parameters in x_t.
- w_t is the vector containing the process noise terms for each parameter in the state vector x_t.
- Q_t covariance matrix for the vector w_t. This covariance matrix represents the uncertainty of the process for each state parameter. The matrix represents how much each parameter
- H_t is the observation matrix that maps the effect of the observations on each parameter in Z_t has on the parameters in the state vector x_t.
- R_t is the covariance matrix for the incoming observations. It represents the uncertainty of each observation. This models how much we should believe the different observations. The observation stream from the systems sensor each have independent uncertainty that needs to be modelled correctly to exploit their properties in the best way.

- v_t is the vector the contains the observation noise for each parameter in the observation vector Z_t. The observation noise is assumed to be zero mean Gaussian noise with covariance R_t.

The Kalman filter algorithm operates in two steps.

Step 1 State prediction

$$\hat{x}_{t|t-1} = A\hat{x}_{t|t-1} + B_t u_t \tag{2.6}$$

$$P_{t|t-1} = A_t P_{t|t-1} + A^T + Q \tag{2.7}$$

Step 2 State correction with observations

This step updates state estimate for $x_{t|t}$ with observation vector Z_t

$$\hat{x}_{t|t} = x_{t|t-1} + K_t(Z_t - H\hat{x}_{t|t-1}) \tag{2.8}$$

Update the error covariance / Project the error covariance ahead

$$P_{t|t-1} = P_{t|t-1} - KHP_{t|t-1} \tag{2.9}$$

Compute the Kalman gain

$$K_t = P_{t|t-1}H^T(H_t P_{t|t-1}H_t^T + R_t)^{-1} \tag{2.10}$$

A full derivations of the these equations is out of the scope of this thesis. A key factor to notice in this algorithm is the Kalman gain. The larger the state covariance is the more we trust our measurements and the larger the K_t is.

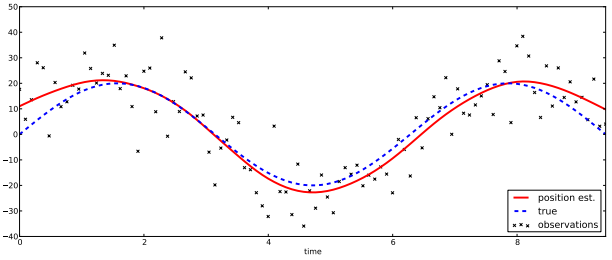

Figure 2.3: In this figure we show Kalman filter for state estimation. We have generated a sine wave were the observation points are subjected to random white noise. The scattered points represent the sine waves data points with the random white noise as an offset and the red line shows are the position estimated by the Kalman filter we applied to the generated dataset. The sine wave without noise is plotted as a blue dashed line.

Outliers

An outlier is generally defined as an observation that "lies outside some overall pattern of distribution" Moore and McCabe [1999]. Outlier may originate from sensor noise, malfunctions and disturbances in the environment. The Kalman filter assumes that all observations lies inside the a Gaussian distribution with a set variance and the performance of the filter will suffer if outliers are present. In Figure 2.4 bellow the a dataset containing outliers originated from a ultra sonic distance sensors mounted on a flying quad copter is plotted. Outlier in this dataset could be identified as observations that lies outside 0.50 meters of the mean. Figure 2.4 shows the quality of the state estimated of the linear kalman filter degrade.

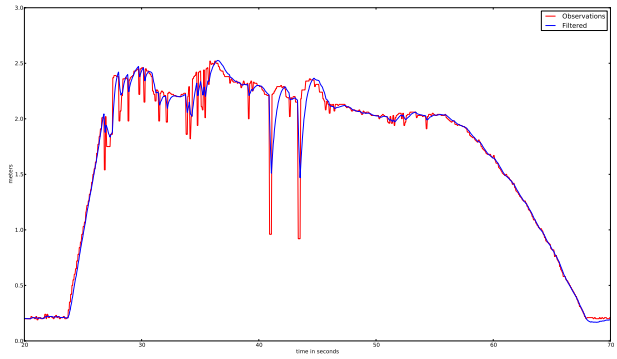

Figure 2.4: This figure shows observations from a ultra sonic distance sensors that contains outliers. The raw observations are plotted as a red graph. The observations filtered by a linear kalman filter are plotted as a blue graph.

Chapter 3

Architecture/Model

In this chapter the hardware and software design of the drone will be presented and explained. In this thesis the system that is airborne will be refereed as the drone

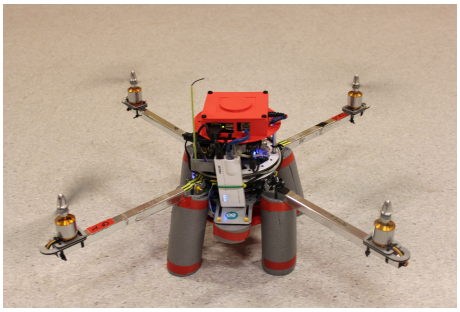

Figure 3.1: The last version of the drone.

3.1 Building the drone

In late January 2013 Artificial Intelligence Group at IDI NTNU purchased two quad copter kits from the online store at www.aeroquadstore.com. The kits cost around 500 dollars each and contain the necessary parts for building a functional quad copter.

List of parts

- Aeroquad v2.1 Flight Control Board kit
- Cyclone Frame kit (arms and chassis)
- 4 x A2217-9 Brushless outrunner motors.
- 4 x 30 Amp Electronic Speed Controllers (ESC's).
- 4 x 10 x 4.7 propellers.

The store sells parts and equipment aimed for the open source community around the website Aeroquad [2013]. The website is the host for the AeroQuad project. The projcet is an open source hardware and software project dedicated to the construction of remote controlled helicopters. The site contains forums and wikis to support building and flying a multicopter.

3.1.1 Building process

The drones were assembled during a two day long workshop at IDI-NTNU. The frame was put together with the electronics. For the sensor shield the sensor needed to be hand soldered to their mounting holes.

First connector pins were soldered to the sensors for stable and easier mounting on the shield. Each solider point was quality checked with a microscope. The best practice suggested in the AeroQuad wikis was followed under the soldering and frame construction

3.2 Hardware platform

The drone is designed with a two processor architecture, one low level processor and one high level processor. The low level processor handles real-time tasks like sensors observations and motor control. The high level processor handles computer vision, filtering, altitude and position control. The low level unit is a Arduino mega 2650 (3.2.2) with a sensors shield (**??**) mounted on top. The Arduino controls the drones motors with four electronic speed controllers (ECS) connected through the sensor shield. The high level unit is a BeagleBoard-XM (3.2).

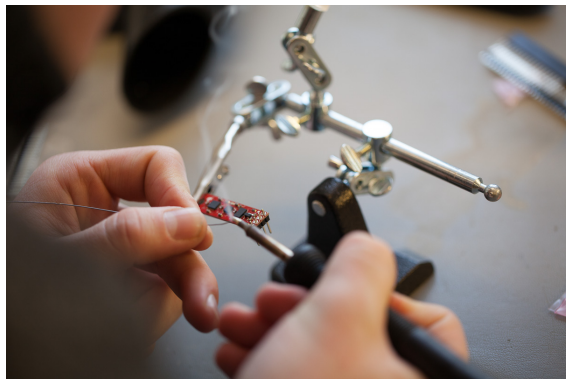

Figure 3.2: Soldering connector pins on 9 DOF sensor

The Arduino and BealgeBoard is connected with a USB cable and communicates through a serial interface 3.6.4. A web camera (3.3.4) is connected to the Beagleboard to enable computer vision tasks and a small and light weight wireless router is connected to enable two-way telemetry **??** and streaming abilities. The Figure 3.2 schematically describes of the hardware parts are connected.

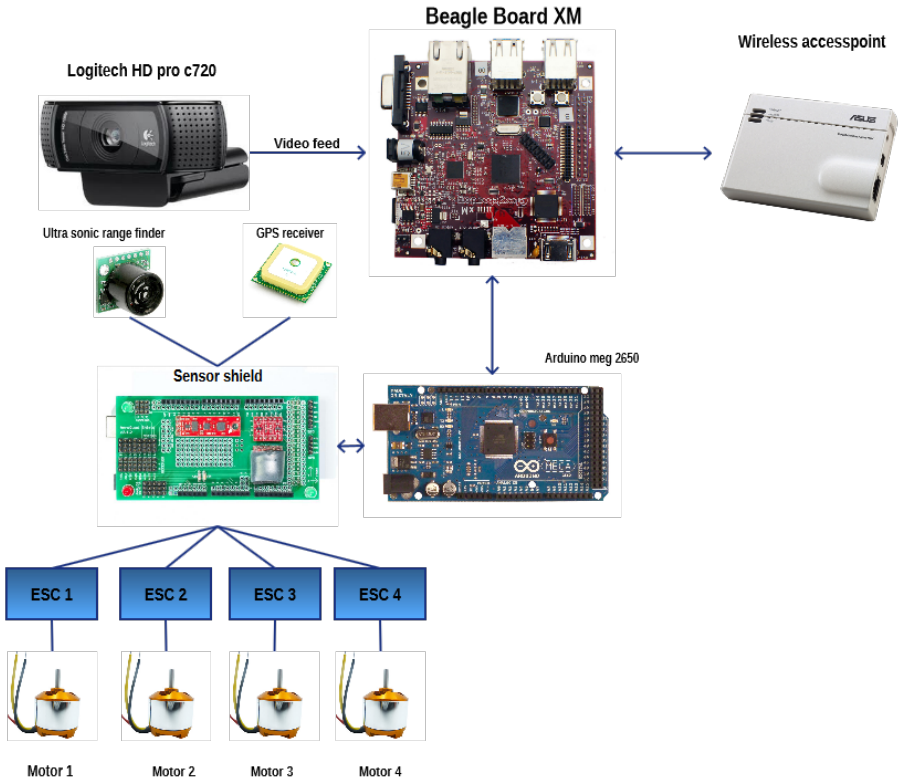

Figure 3.3: This figure shows how the different hardware parts that forms the drones autonomous navigation system is connected.

3.2.1 BeagleBoard-XM

The BeagleBoard-XM is a low powered hardware embedded computer based on the ARM37x 1 GHZ Cortex-A processor. For this thesis the board used for the high level computations, computer vision, senor fusion, streaming, telemetry and navigation. The board was mounted on top of the drone with a 3D-printed mounting part (3.4.1). The board ran an ARM based verison of Ubuntu Linux 12.04.

CPU	ARM Cortex A8 1 GHZ
RAM	512 LPDDR
USB	4 x USB 2.0
Network	100 Mbit Ethernet
Size	85 mm x 85 mm
Weight	37 g

Table 3.1: Highlighted specifications for the BeagleBoard-XM

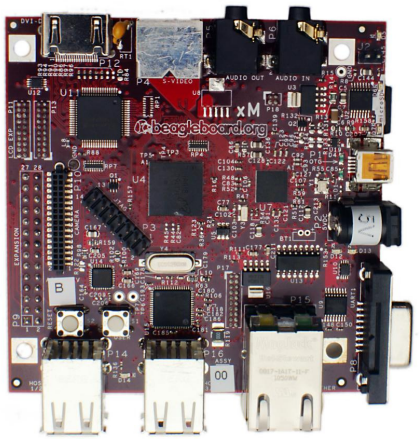

Figure 3.4: Beagle board XM

3.2.2 Arduino 2560 Mega

The Arduino 2560 is a micro controller board based on the ATmega2670. It has a number of digital and analog input and output pins and a USB connection for serial interfacing. For this thesis the Arduino was used to gather the data from the sensors mounted on the sensor shield. The sensors shield is mounted on top of the Arduino making input pins for motors and external sensors easily accessible and holds the stationary sensors in place. The data from the sensors are processed by the flight software and keeps the drone in stable flight.

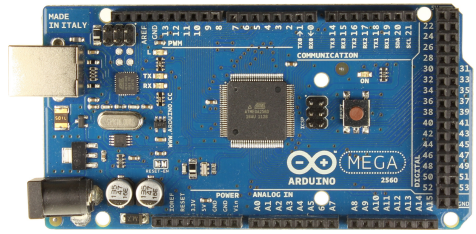

Figure 3.5: Arduino mega 2560

Sensor shield

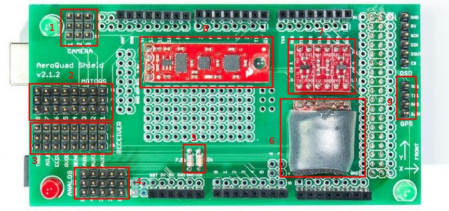

Figure 3.6: Aeroquad Sensor shield v2.1

1	Three camera servo inputs
2	Motor outputs
3	Receiver input channels
4	Analog inputs, Used for Sonar sensor
5	Resistors for batteryMontioring
6	Barometric sensor
7	Logic Level Translator
8	9DOF IMU, Accelerometer, gyroscope and magnetometer.
9	GPS input pins

Table 3.2: Description of components on the sensor shield

3.2.3 Telemetry

The telemetry is the connection between the drone and the ground station. The connection was done over a portable router. An Asus WL-330N mounted on the drone and connected to the Beagleboard through its Ethernet port connects the drone and the ground station. The reason for this solution is rather than just connecting a WI-FI dongle is that this solution gives better range, bandwidth and spares one USB port and USB-bandwidth.

3.2.4 Radio control

Although the goal for this thesis is to make the drone navigate autonomously it is necessary to have R/C connection to the drone for testing purposes and override under autonomously flight if the autopilot for some reason should go rouge. The R/C is achieved through a Hi-Tec Eclipse 7 8-channel receiver and transmitter. The receiver is mounted on the drone and connected to the sensor shield. There is one signal cable for each channel that maps to the corresponding channel on the shield.

1. Yaw
2. Roll
3. Pitch
4. Throttle
5. Autopilot
6. Altitude Hold
7. Attitude mode (rate / attitude)

Override

The override is a switch on the transmitter that enables fast switching between manual and autonomous flight. This is useful when testing out new autonomous features on the drone and getting manual control back as fast as possible if needed. To enable this feature the source code of the AeroQuad was modified. The original source code reads the incoming receiver channels on the Arduino in a for loop where there is added a function to check if the auto switch is turned on or off. Functionality is also added so it is possible to select which channels (3.2.4) that should be under autonomous control. This is useful because it makes us able to independently run autonomous control on one or multiple control channels.

3.2.5 Hardware costs

Hardware costs are one of the research questions in this thesis (1.2). The bill of matériels are presented bellow.

Component	Cost
AeroQuad Cyclone ARF Kit	$535
Ultra sonic sensor	$50
Battery 4000 mAh 3 cells	$20
BealgeBoard-XM	$199
Logitech HD Pro Webcam C920	$100
Asus WL-330N	$35
Total	$939

Table 3.3: Bill of materials

3.3 Sensors

3.3.1 Barometric sensor

A barometric sensor measures the atmospheric pressure above sea-level. From the measured pressure the absolute altitude can be calculated from the international barometric formula.

$$altitude = 44330 \times \left(1 - \left(\frac{p}{p0}\right)^{\frac{1}{5,255}}\right) \tag{3.1}$$

Where p is the measured pressure in and p0 is the pressure at sea level. We can calculate relative altitude by initializing a ground altitude before take off.

$$relativealtitude = groundaltitude - altitude \tag{3.2}$$

As for all sensors the barometric has its limitations. It is sensitive to changes in the weather in particular wind and temperature that could lead to drop or rise pressure. The pressure will then be different from when the sensor was initialized and the altitude will suffer from a offset.

3.3.2 Ultra sonic range sensor

An ultra sonic range sensor emits a high frequency beam of sound waves and interpreter the echoes. The time interval between sending the waves and receiving

the echo is used to calculate the distance to an object. The ultra sonic sensor used in this thesis is the XL-Maxsonar MB1200 from the supplier Maxbotix. The mounting of the ultra sonic sensor is essential for its performance. A quad copter is a challenging environment to mount a range sensor. There are multiple sources of noise and bias on the airborne drone as described in Gross [2013].

- Air turbulence from the propellers wash.
- Propeller acoustic noise
- Conducted and electrical noise on the shared power supply.
- Radiated electrical noise from other electrical components
- Frame vibration

Limitations and issues with the ultra sonic sensor

- Resolution at 1 cm
- Can produce non-Gaussian noise.
- Minimum range at about 30 centimetres.
- Maximum range at about 6 meters

3.3.3 Inertial measurement unit

The inertial measurement unit (IMU) is a collection of sensors. In this build the the IMU has 9 degrees of freedom obtained by an accelerometer, gyroscope and magnetometer.

- The Accelerometer measures the acceleration in 3-axis. x, y and z.
- The Gyroscope measures the angle in 3-axis. x, y and z.
- The Magnetometer measures the compass direction in 3-axis. x, y and z.

The sensors on the IMU have some limitations and bias.

- The accelerometer drifts and accumulates error over time.
- The gyroscope accumulates error over time.
- The magnetometer accumulates error over time x, y and z

These limitations and biases was found during experiments conducted in thesis.

3.3.4 Camera

The camera is a Logitech HD Pro C930e webcamera. The camera can produce video streams in three formats RAW, MJPEG and H264 and in a variety of resolutions from 90p to 1080p and has a 90 degrees field of view. The camera was connected to the Bealgeboard using USB 2.0.

Figure 3.7: webcamera used for this thesis

3.3.5 Camera calibration

The distance to a physical object can be calculated with a monocular camera can be calculated with this formula3.3.

$$Z = \left(\frac{D \times f}{d} \right) \tag{3.3}$$

Where the

- Z is the distance to the object
- D is physical size of the object
- d is size of the object in pixels
- f is the focal length of the camera in pixels.

The focal length in pixels was measured by holding the camera in a fixed position in front of the marker. The marker size, distance to the marker was measured and the focal length could be calculated with the same formula (3.3) with respect to f.

3.3.6 Hardware issues

When dealing with hardware and multiple hardware devices that are working together there are a variety of issues could arise.In this thesis a hardware failure will be critical and potentially dangerous since it could lead to the drone losing control in mid air and crashing. The hardware was carefully tested before mounted and connected on the drone build, but not everything can be foreseen and tested.

Hardware failures

In this thesis we have experience a wide variety of hardware failures that have led to crashes. One of the major incident that led to hard crash of the drone was radio antenna mounted incorrectly. In the first outdoor test of the drone it had been 20 seconds in the air before it crashed. The drone broke three propellers, motor mounts and was bend out of shape. From the sound of the motors the it sounded like malfunction on one or more of the ESC‘s as one or more of the motors spiked in thrust making the drone flip and spin into the ground. Back on the lab troubleshooting began, first on the ESCs and later on the different sensors. Nothing wrong could be found. After some troubling hours with debugging it was pinned down the problem existed some were between the transmitter, receiver or receiver and sensor shield. Multiple attempts to recreate the malfunction were done, carefully monitoring every data and sensor output from the drone. After multiple sessions it was discovered that the way the antenna on the receiver was mounted was closely linked to how much interference it picked up. For practical reasons the antenna on the drone had been lashed around the chassis of in a circular manner because of length. This a lot of interference as shown i Figure 3.8, when the transmitter control was not pointed directly on the drone or the transmitter was turned of.

In conclusion the drone crashed was caused by high interference on the receiver due to incorrect mounting of the antenna and transmitter not directly pointed at the drone in flight.

This is one example a hardware failure that caused the drone to crash, throughout this thesis we have summed up four hard crashes and one fire, that led to serious damage to the drone. We have almost changed every part on the drone at least one time from damage that came from crashes and parts that just stopped working. One of the interesting things is that the drones crashes strictly came from hardware failure like antenna mounting and power cables that melted in mid air and not from rouge autopilot as we feared in the beginning. In figure 3.9 the drone is crashing to the ground due to a power malfunction and in the Video (http://www.youtube.com/watch?v=Io98jnSxHA8) the drone crashed due to a melted power cable (at least it hits the marker).

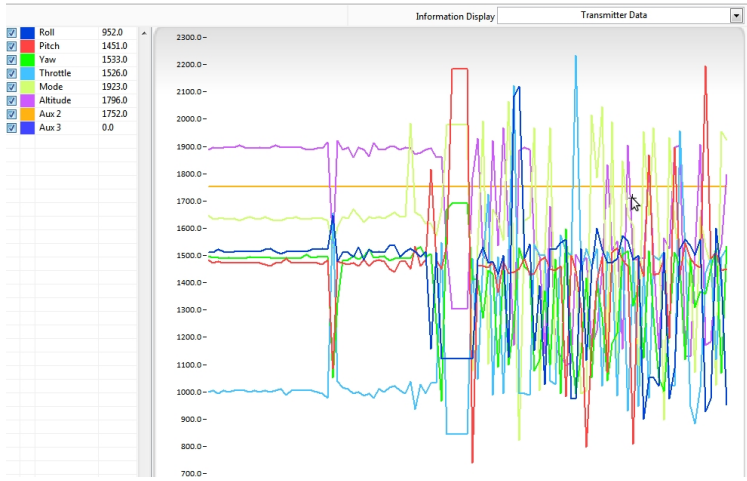

Figure 3.8: This figure shows the drone receiving heavy noise over the radio control channels.

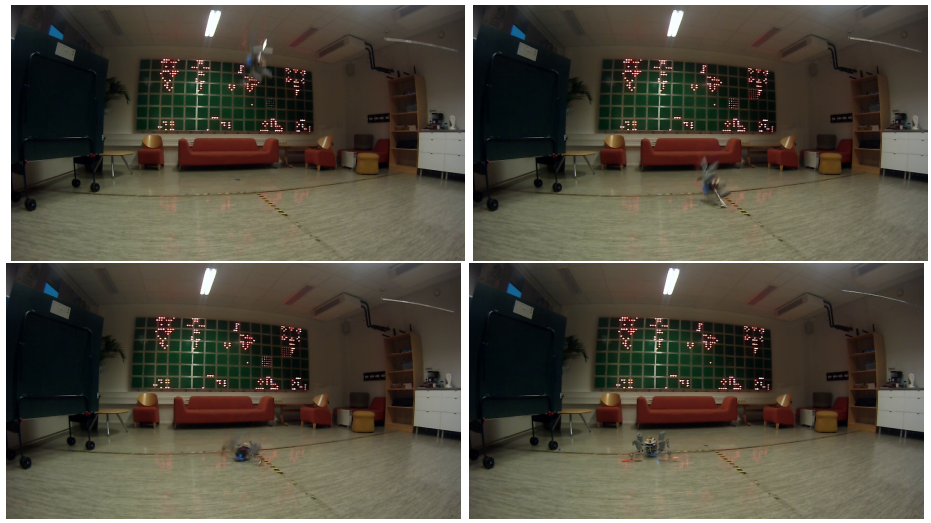

Figure 3.9: We can see the drone loosing its power about 2.5 meters in the air and causing it crash into the floor.

3.4 Designing parts

In this section I will describe how I designed and printed parts to the drone. For this thesis on-board computing is an essential part. The hardware required to make this possible is needed to be mounted on the drone. There is no commercial product available for mounting this kind of hardware on this type of drones. The hardware needed to be stable and safely mounted on the drone for optimal performance and protection. The drone operates in harsh working environment with contentious hard landings and crashes. Crashes are expensive and very time consuming. Repairing the the drone and waiting for the delivery of replacement parts often take days and possibly weeks if parts need to be sent from foreign suppliers. The strategy of protecting the critical parts with replaceable parts that could be printed on the lab was applied. The parts needed to be designed so the ratio of robustness and weight was suitable for a flying robot. The printing was done on a 3D printer Makerbot Replicator 2. This printer makes it possible print plastic parts designed in a 3D-modelling software (Autodesk Invetor). The printer uses a plastic-filament and the printing software lets you customize the percentage of infill in your parts. This enables you to find the right combination of robust and lightness in the drone parts 3.10.

3.4.1 BeagleBoard mount

In the need for safe and stable mounting for the BeagleBoard a mounting device for the drone was designed. The device should protect the BeagleBoard in crashed and hard landings, mounted on the drone with screws for stability and be as light weight as possible to obtain the manoeuvrability of the drone. The device is fitted on the drone on top of its exiting layer architecture.

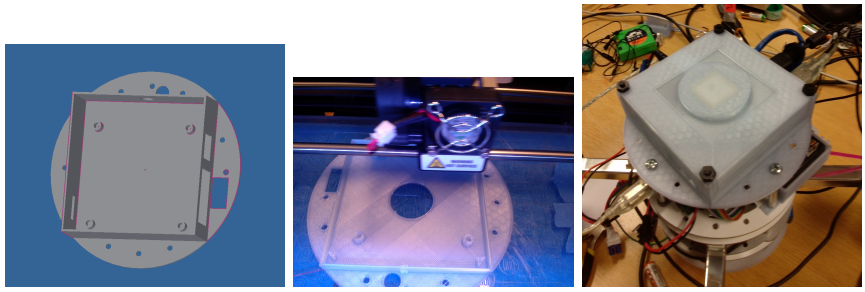

Figure 3.10: This figure shows the production line of the beagle board mount. In left image is a screen shot of the blueprint for part. The middle image shows the part being printed in plastic. The last image shows part mounted on the drone with the Beagleboard Figure (3.4) inside and a protection lid on top.

3.4.2 Sensor module plate

To mount the camera and the ultra sonic sensor a sensors module plate was designed to fit these at the bottom of the drones body.

3.4.3 Camera mount

The camera mount was designed to fit the sensor plate (3.4.2). The camera mount holds the camera in stable fixed position and the lens pointed downwards in the vertical direction. The web camera was stripped down removing most of its plastic body and stand to its essentials keeping the wiring and some of its body to protect the electronic components. This was to make it as light possible and fit in the designed camera mount.

3.4.4 Ultra sonic sensor mount

To ensure the performance of the ultra sonic sensor the mount point needed to be as far from the ECSs as possible because of the electronic radiation they cause and as near the body frame center to avoid turbulence from the four propellers of the drone. The measurements taken to ensure the mounting of the ultra sonic sensor as discussed in section 3.3.2. A mount was designed to fit the ultra sonic sensor and the so the mounting holes on the sensor plate (3.4.2).

3.4.5 Propeller guards

Flying and operating a drone can be potentially dangerous. The greatest risk is the spinning propellers, if it interacts with living person it could cause serious injuries. In this thesis there have been taken precautions to protect our self from injury from the drone. The process of designing the propeller guards a great deal of ideas was taken under consideration. What seemed to be the most reasonable were to design the guards using a 3D modelling software and printing them with a 3D printer, the Makerbot 2.0. The prototyping started using Google SketchUp. The first prototype was heavily inspired by the guards on the Aeroquad Tyfoon beta frame show in Figure 3.11.

Figure 3.11: Inspiration for the propeller guards

We started out printing my prototypes early to get a feel for the properties of the material and prototypes construction. After a number of trying and failing a "fly-ready" prototype was tested. The test did not go as expected when the drone was experience semi-hard landings the guards bounced up into the propellers causing them to explode on impact making actually making the drone more dangerous, shooting plastic shrapnel around. This prototype was shelved. A new prototype was designed from scratch using a more advanced 3D modelling software (AutoDesk inventor) to cope with the limitations of Google Sketch up. Making the 3D model more modal and easier to work with. The new propeller guard contains of two parts or optional third part for full protection. The main reason for this is that the 3D printer that was used simply can not print the hole guard in one session due to size limitations. There are some pros and cons to take under consideration when printing. Its important to find the right combination of strength and lightness. In Figure 3.12

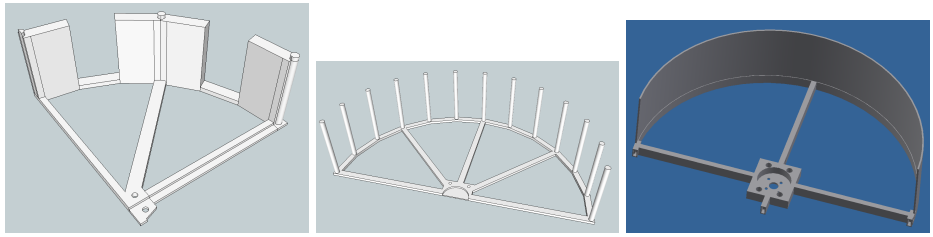

Figure 3.12: Evolution of propeller guards

3.5 Software platform

The system of the running the drone is a series of software working together The software runs on two different platforms. The BeagleBoard 3.2 runs the high level computations, such as image processing, position controller, Kalman filtering and logging and the Arduino runs the low level operations controlling the attitude of the drone and keeping it in the stable flight. In figure 3.7 is a schematic description of how the applications is dived.

3.6 BeagleBoard software

The software that is deployed on the BeagleBoard is designed to capture the video feed from the web camera, read sensor data from the Arduino (3.2.2) through serial interface, filter them and navigate the drone to its desired state or destination. In this thesis the main objective is to identify the drone landing spot, fly there and land. All the computing necessary to solve this problem is done on-board and the drone must be able to act independently without interference from any ground station or radio transmitter. For debugging and observability causes the system is designed to stream live video back to the ground station (3.6.3). The software on running on the BeagleBoard and the ground station software is developed from scratch and written in Python. It uses collection of frameworks such as OpenCV, PyKalman and Gstreamer.

3.6.1 Video capture

The web camera is connected to the BeagleBoard using a USB 2.0 connection and captured with Gstreamer using the Video 4 Linux driver. Gstreamer is a pipeline-based multimedia framework that lets you customize and build a pipeline with different media elements. In this application this is useful because it lets us choose different parameters for the video stream from the web camera, such as the frame rate, image resolution and vide codec. For image recognition and target tracking these parameters are important. The higher the frame rate is, the more accurate the tracking are, on the other hand the processing power in this system is limited and it is powered by battery. A lower frame rate will need less processing power, but could lead to more inaccurate tracking. The image resolution also has a impact on the processing power: the higher the resolution, the higher the computational costs for each image processed is. These parameters eventually depend on cameras abilities to deliver them. In this thesis the Logitech c930e was used. This is one of the web cameras on the market that gives the widest variety of options for these parameters as described in Section 3.3.4. The media pipeline is schematically described in Figure 3.13.

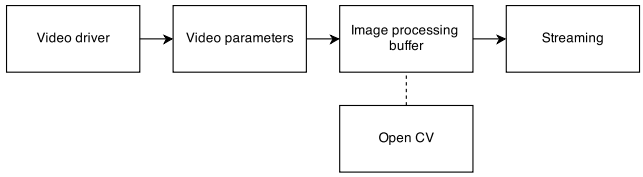

Figure 3.13: The video driver element captures the frames from the camera. The video parameters element sets the codec, frame rate and resolution for the video driver element to capture. The Image processing element sends a copy of the frame passing through the pipeline to the image processing model. The streaming element streams the video feed over the wireless network to the ground station. The feed is streamed over UDP and encapsulated in RTP.

3.6.2 Computer Vision

For this thesis open source computer vision library Open CV is applied to recognize the landing spot.The landing spot will act as an marker. The maker is recognized through color and shape. For the marker we have chosen NTNU‘s logo and alternated its color to red. This is to make it more distinct in environment and easier to recognize. Open CV offers a great deal of functionality to help sort out color and shape in a given image. Below we describe the algorithm designed in steps to make this possible.

First the incoming frame from the web camera is converted from BGR colorspace to the HSV (Hue, Saturation, Value) colorspace. The reason for this is that it is easier to recognize a color in this colorspace.

- H , Hue represents the color
- S , saturation is the amount of color that given color
- V, value is the how bright the color is

The marker color is represented in an array that have the values of the H ,S and V for the color to track. The OpenCV function inRange() replaces the pixels within the threshold of the target color with white and the rest with black. This gives a binary image where the areas with the target color is coloured white and background is coloured black. Contours can easier be identified from the binary image. The areal of the contours are calculated by counting the number of pixels inside the contour, the contour with the greatest areal is selected to most

likely be the target marker. Contour approximation is applied to the contour to smooth out defects caused by small curves in the contour. The OpenCV function moments() calculates the image moment for the given contour. From the moment we could get the pixel position of the contours center in X and Y direction and this is what we need to estimate the horizontal distance in pixels to the marker.

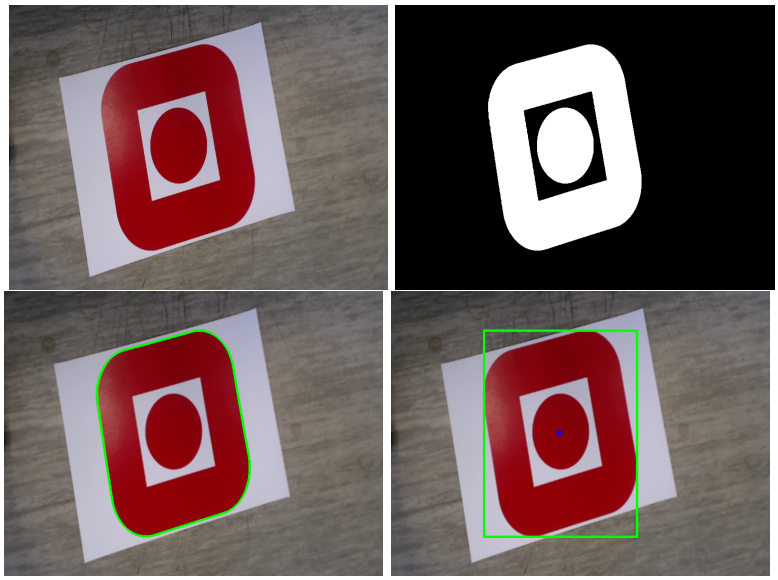

Figure 3.14: The images visualize the transformation the captured video frames goes through in 3.6.2 from top left to bottom right. The first image are the raw captured frame, in the next image thresholding on the target color is applied. In the third picture the largest contour from the thresholded image is drawn on the raw image. In the last image the contour center and contour approximation are visualized

3.6.3 Ground station

The ground station was used to monitor the drone under flight, debugging and tuning parameters on location when experiments were conducted. The ground station was a laptop running software that connected to the live video feed from the drones camera and showed it images in real time and visualised the marker recognition on the screen. In Figure 3.6.3 the ground station in action.The video was saved on the laptop for later analyses. The software for the ground station was developed from scratch for this thesis with the support of third party libraries

mentioned in section 3.6.

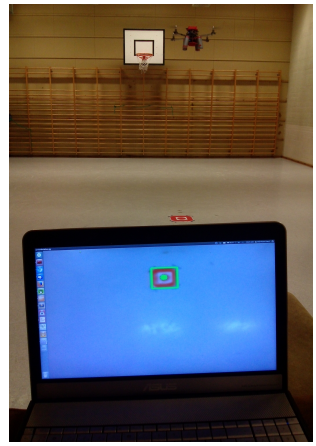

Figure 3.15: This figures shows the ground station software (3.6.3) deployed on a laptop. The laptop is connected to the drone through Wi-Fi (3.2.3) and are receiving a live video. The ground station presents the video feed and visualizes the computer vision algorithms (3.6.2) deployed on the drone.

3.6.4 Interfacing

The interfacing between the Arduino and the Beagle Board is done by USB serial connection running at an baudrate of 115200. The AeroQuad software have support for serial communications for intended for configurations of the flight software. For this thesis we will use this communication channel to interface with the flight software on the Arduino. In a two-way communication where the Arduino sends the measured sensor data to the BeagleBoard and reads the controller commands u sent from BeagleBoard and the position controller. The sensor measurements are sent at 20 HZ and the control commands are sent at 50 HZ. The communication is schematic described in 3.6.4

Limitations

Since the serial communication included in the AeroQuad-software is intended to be used to purely for on the ground configurations of the drone. Thsere are some limitations with the software. The sensor data sent is over serial is sent as a comma separated string this is quite CPU intensive on the sender and receiver side when the sample frequency gets high. Higher sampling frequency in the

sensors measurements and control commands will lead to less delay and more responsive and accurate system, this is futher discussed in the the section **??**. The robustness of sending a the sensor data over serial in clear text can vary and where there is no checksum used.

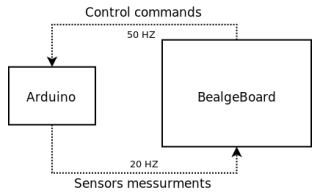

Figure 3.16: 2 way communication between the Arduino and the Beaglboard

3.7 State estimation

For an autonomous drone to perform well in real-world applications it needs to have some perception of the world it acts in. Since the world state can not be observed directly the drone needs some kind of estimate of the state. The estimate is obtained through observations from the drones sensors and updated continuously according to new observations and actions using the different models. The observation model (3.7.3) describes how the sensor observations is linked to state estimate and the motion model describes how the state changes according to the control commands issued and the drones dynamics. To give this estimate of the world and update it continuously the Kalman filter is implemented.

3.7.1 The state space

The internal state of Kalman filter is defined as a vector of state parameters

$$\hat{X} = [\bar{x}, \bar{y}, \bar{z}, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z}, \Phi, \Theta, \Psi] \in R^{12} \tag{3.4}$$

Where

- $\bar{x}$ and $\bar{y}$ are the drones position in meters relative to the landing spot.
- $\bar{z}$ is the altitude of the drone in meters.
- $\dot{x}, \dot{y}, \dot{z}$ are the velocity of the drone in meter per second.
- $\ddot{x}, \ddot{y}, \ddot{z}$ are the accelerations of the drone in meters per second.

- Φ, Θ, Ψ are the roll, pitch and yaw Euler angles.

The state changes over time and are updated 20 times in one second. The state gets updated according to the motion and observation model.

3.7.2 Motion model

The motion model defines how the dynamics of the drone and the control commands influence the predicted world state. We could help the Kalman filter with its prediction by defining a good motion model.

Hovering

The drone hovers when the vertical acceleration is at zero and the drone is in flight. Gravity is pulling the drone with a downward force and the motors are keeping it afloat. The vertical acceleration is given by the Equation 3.5.

$$m\ddot{z} = F - mg \tag{3.5}$$

Where F is the sum of forces gained by the motors $F = F1 + F2 + F3 + F4$ and m is the mass and g the gravitational acceleration. Rather than calculating the force each motors generates this models uses the battery voltage to estimate the thrust needed to keep the drone hovering. The amount of thrust needed to hover increases through out the flight time and is dependent on the battery voltage that decreases in a linear fashion.

Horizontal velocity

To calculate the horizontal velocity in x and y direction the attitude of the drone is calculate in 3.6 and 3.7.

$$\dot{x} = c_1(\cos\Psi \sin\Phi \cos\Theta - \sin\Psi \sin\Theta) \tag{3.6}$$

$$\dot{y} = c_1(-\sin\Psi \sin\Phi \cos\Theta - \cos\Psi \sin\Theta) \tag{3.7}$$

where

- Φ, Θ and Ψ is the roll, pitch and yaw angle in Euler angles
- c_1 is a constant found through a series of test flights.

Control commands

The control commands are the commands sent to the on-board flight software and tells the drone how to act. The control commands are defined in Vector 3.8.

$$u = [u_\Phi, u_\Theta, u_\Psi, u_{\dot{z}}] \tag{3.8}$$

The control commands originally ranges from 1000 to 2000, but are normalized in the range $[-1, 1]$ before fed into the Kalman filter. The Kalman filter has support for control inputs as shown in the prediction Equation 3.9

$$\hat{x}_t = A_t x_{t-1} + B_t u_t + w_t \tag{3.9}$$

The Kalman filter uses the control input to improve its prediction when the observations are missing or to biased. In this application the control commands are used when the drone loses track of the landing marker. Where u_t is the control input vector and B_t the control matrix. The control inputs are caluated in the Equations 3.10, 3.11, 3.12 and 3.13

$$\ddot{u}_\Phi = c_2(u_{\Phi t} - u_{\Phi t-1}) \tag{3.10}$$

$$\ddot{u}_\Theta = c_2(u_{\Theta t} - u_{\Theta t-1}) \tag{3.11}$$

$$\ddot{u}_\Psi = c_2(u_{\Psi t} - u_{\Psi t-1}) \tag{3.12}$$

$$\ddot{u}_z = c_2(u_{zt} - u_{z_t-1}) \tag{3.13}$$

where

- $u_{\Phi t}$, $u_{\Theta t}$, $u_{\Psi t}$ and $u_{\Psi t}$ is the current control input
- $u_{\Theta t-1}$, $u_{\Psi t-1}$, u_{zt} and $u_{\Phi t-1}$ is the previous control input
- c_2 is a constant found through a series of test flights.

The control commands give an estimate of the acceleration of horizontal acceleration in x and y direction and are fused with the parameters $\ddot{x}$ and $\ddot{y}$ in the update filter step.

3.7.3 Observation model

In the correction step of the Kalman filter algorithm the observations from multiple sensors are fed into the process updating the state and giving us a better estimate of the state. By combing and fusing the sensors observations together the key idea is that the sensors will compensate each other biases. The observation model describes the mapping for the observation matrix H and the sensor covariance matrix R. The observation vector Z represents the stream of sensor observations fed into the Kalman filter.

Measuring horizontal distance

To convert the distance to the marker in pixels obtained from the algorithm described in Section 3.6.2 to meters we can use focal length found in Section 3.3.5. Since the drone has an attitude when in motion and the camera was mounted in a fixed position at drones rigid body the camera has an attitude as well. To compensate for the offset caused by an attitude greater than zero the measurement from the gyroscope was used. The Equations 3.15 and **??** makes two assumptions.

- The camera is mounted in level with the gyroscope.
- The distance $\bar{z}$ equals the distance in the vertical line to the ground and is not affected by the attitude because of the width of the ultra sonic sensors as beam described in Section 3.3.2.

$$x = sin(\Phi)\bar{z} - \Delta p_y \frac{f}{\bar{z}} \tag{3.14}$$

$$z = sin(\Theta)\bar{z} - \Delta p_y \frac{f}{\bar{z}} \tag{3.15}$$

where

- x and y is horizontal distance to the marker in meters.
- f is the focal length of the camera
- Δp_x is difference in pixels between the camera width center and the markers center point.
- Δp_y is difference in pixels between the camera height center and the markers center point.
- Φ and Θ is the roll and pitch Euler angle

Observation vector

The vector for the observation parameters is described in Equation 3.16.

$$Z = \begin{bmatrix} x, y, \dot{x}, \dot{y}, z, \Phi, \Theta, \Psi \end{bmatrix} \tag{3.16}$$

where the

- x and y is the distance to the marker in x and y direction found in Section 3.7.3.
- z is the distance measured by the ultra sonic distance sensor.
- $\dot{x}$ is the velocity in x direction calculated in Equation 3.6.
- $\dot{y}$ is the velocity in z direction calculated in Equation 3.7.
- Θ, Φ, Ψ Roll, pitch as Euler angles obtained for gyroscope.

Each of parameters in the observation vector Z has an independent covariance represented in a covariance matrix R. The observation covariance represents the uncertainty of the measurements. This enables us to model the noise of each of the sensors and tell the Kalman filter how much it should believe these measurements. The covariances of the sensors is found through experimental testing and running the Kalman filter on offline data collected with throughout flight tests.

Handling outliers

An outliers is a noisy sensors measurement that lies outside the Gaussian distribution. The outlines need to be identified and handled. In this implementation the ultra sonic distance sensor is suffers from this kind of non-Gaussian noise. To handle this the measurements gets checked for outliers before the kalman filtering. If the change between the an incoming measurement and the average value of the ten last measurements is greater than 0.5 meters the measurements are identified as an outliers. The kalman filter then threats this an missing measurement and predicts the new state using the motion model and process model.

3.7.4 Process model

The process model consists the state transition matrix A and its covariance matrix Q. The state transition matrix models the linear transformation for the state $\hat{x}_{t-1}$ goes through to get to $\hat{X}_t$. In the state transition matrix we have implemented the equations of motion and they are defined as Equation 3.17 and 3.18

$$x_t = x_{t-1} + \dot{x}_{t-1}\Delta t + 0.5\ddot{x}_{t-1}\Delta t^2 \tag{3.17}$$

$$y_t = y_{t-1} + \dot{y}_{t-1}\Delta t + 0.5\ddot{y}_{t-1}\Delta t^2 \tag{3.18}$$

From these equations we can estimate the position parameters x_t and y_t for the current state $\hat{X}_t$, transformed from the parameters in the previous state $\hat{X}_{t-1}$. This enables us to give a prediction of the the state $\hat{X}_t$ that uses the previous velocity and acceleration and give a estimated based on this transformation is important when dealing with missing observations. The process noise is defined in the covariance matrix Q. It defines the uncertainty of the parameters in the process. The covariance for these parameters was found by experimental research using offline data collected through a series of flight. The data was both video from the web camera and logged sensor data. The covariance used for the process is presented in the Matrix 3.19

$$Q = \begin{pmatrix} 0.001 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.001 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.001 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.001 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.001 \end{pmatrix} \tag{3.19}$$

3.7.5 PID controller

This drone have multiple PID controllers and for its different applications. This is further discussed in the descriptions of the different applications. The PID controllers are responsible for giving correction to the control input so the application can reach its set point. The PID controllers have been designed as described in Section 2.2.2.

3.7.6 Altitude hold

In altitude hold the drone tries hold an altitude over time as stable as possible. When the altitude hold is enabled by the user, the altitude at the given time is

set to a initial set point of the PID controller. The current throttle and battery power are set as initial values. The PID controller applies a correction to the initial throttle based on the present, past and predicted error based on the P, ID parameters set. This will work for some time, but after some time the drone slowly loose altitude since the amount of force given by the motors is not constant and degrades as the battery drains . The constraint of the maximum and the minimum correction that could be given inebriates the drone to hold its desired altitude. To cope with this the battery voltage is taken into account in the as shown in Equation 3.20

$$u_z = \gamma + T_0 \frac{B_0}{B_t} \tag{3.20}$$

Where

- u_z is the new throttle with corrections.
- γ is the correction from the PID controller.
- T_0 is the initial throttle.
- B_0 is the initial battery voltage.
- B_t is the battery voltage at time t.

3.7.7 Position hold

The objective in position hold is to hold a position in roll and pitch direction stable over a period of time stable. To enable this to happen the drone needs to have a some measurement of its whereabouts. In this thesis we use the state parameters x and y from the estimated state $\hat{X}$ from the Kalman filter described in Section 3.7.4.The roll and pitch axis has its own independent PID controller and the set points to the controller are set to 0.0, in a optimality solution this would place the drone directly above the marker. The PID controllers calculates a correction for the control input u_Θ and u_Φ and the control commands is issued to the on-board control loop.

3.7.8 Autonomous landing

The autonomous landing is a combination of the altitude hold (3.7.6) and position hold (3.7.6). When the drone have positioned itself in a steady state over the marker. The user can enable the autonomous landing from a dedicate channel on the remote as described in Section 3.2.4. The PID controller for the autonomous landing uses the desired decline velocity as a set point. From the state estimate $\hat{X}$ we can get the vertical velocity from the state state parameter $\dot{z}$. PID controllers calculates a correction for the control input u_z and the the control commands is the on-board control loop. Since the ultra sonic sensor sensors is not able to return accurate measurements under 20 - 30 centimetres the drone goes in for the final stage landing when it reaches this altitude, decreasing the throttle rapidly the at the last stage. To identify when the drone is safely on the ground and the motors can be disarmed we look at the attitude pattern for the last 60 time steps. By looking at the variance we could tell if the drone has landed or not.

Chapter 4

Experiments and Results

4.1 Experimental Plan

The initial experimental plan for this thesis was to iteratively implement features to reach the main research goal of precision autonomous landing described in Section 1.2. The sub goals was approached from top to bottom, with the assumption the prior goal needed to be implemented, tested and stable before the next goal could be addressed. The override switch enabled us to test the sub goals as independently as described in Section 3.2.4.

During thesis experiments was conducted both inside and outdoors. In the following Section 4.2 only results from the indoors experiments is presented, this is further discussed in Section 5.3.

4.2 Experimental Results

In this section the results from the experiments conducted in this thesis will be presented. We present the results from the research sub goal described in section 1.2. The sub goal altitude hold is presented independently section 4.2.1. And the position hold and autonomous landing is presented in section 4.2.2.

4.2.1 Altitude control

In the altitude control experiments the drone was first flown manually to an random altitude between approximatively 1.5 to 3.5 meters and when the drone was in stable position the autonomous control was enabled and the altitude hold implementation described in Section 3.7.6 was applied. We will measure the performance of the drones altitude control in two different experiments.

- The ability to hold a set point.

- The ability to reach a changed set point.

Endurance

In the experiment presented in this section we will look at the drones ability to hold an altitude given by the initiated set point over a period of time. In figure 4.1 shows how the altitude control performed. The the altitude represented by the blue line is to the red line representing the set point. The closer the blue line is to the red, the closer the drone is to its set point. The Table 4.2 show us the drone is able to stay within an average error of 0.13 meters, a variance of 0.15 meters and standard derivation of 0.095 meters over time of 47 seconds. In the PID parameters used in this experiment is presented in Table 4.2. Figure 4.2 shows that the D correction had the most impact on the total correction and is what you will expect when the drone is in a steady state with a low error.

Set point	2.45 meters
Average error	0.13 meter
Variance	0.15 meters
Standard derivation	0.095 meters
Elapsed time	47 seconds

Table 4.1: Statical results from the altitude hold endurance experiment

Kp	25
Ki	0.5
Kd	80

Table 4.2: Values for the PID controller used in Section 4.2.1.

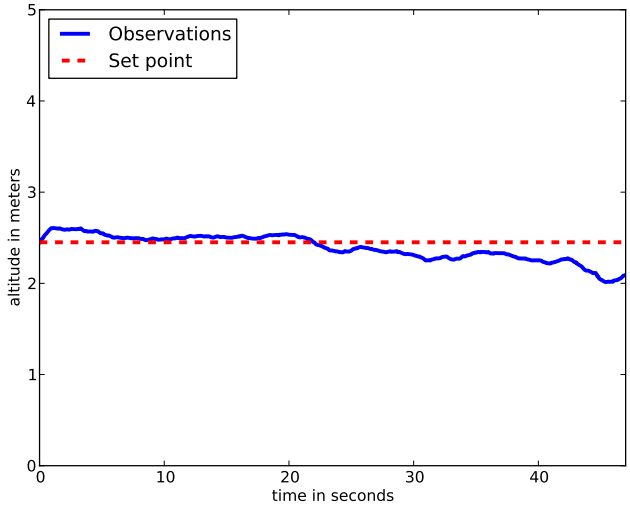

Figure 4.1: The blue line represents the altitude of the drone in meters and the red line dashed line is the altitude its attempt to hold.

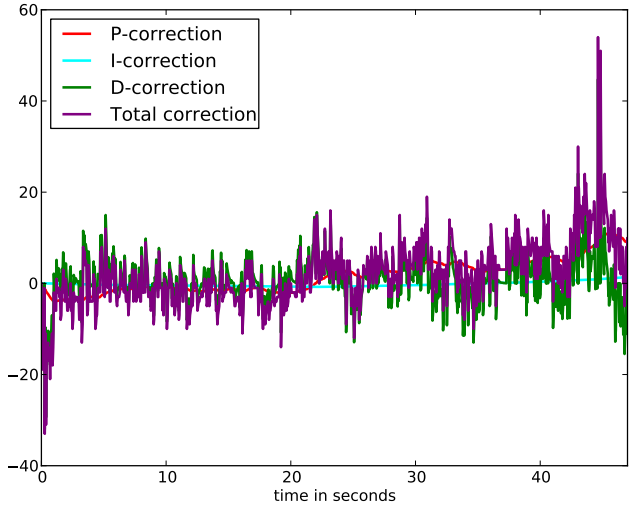

Figure 4.2: This figure shows the PID corretions issued to the control command u_z under the experiment.

Altitude trajectory

In this experiment we tested the drones ability to go from one altitude to another. The drone was first manually flown to an altitude to about 1.5 meters before the autonomous control was enabled. The drone then tried to hold initial set point of 1.5 meters in 8.7 seconds until we manually altered the set point while the drone was in flight. The new set point was set to the initial set point and 1 meter was added.

In Figure 4.3 shows how the drone is trying to hold the altitude in the time period until the set point was altered. After the set point is altered we we can see the P corrections shots up causing the drone to accelerate in a positive vertical direction, the D compensates for the P corrections giving a negative value to prevent the drone to overshot its set point. The I corrections of have almost no impact on the total correction. The PID controller is in no danger of overshooting its set point and it settles around 0.40 meters bellow the set point and this stagnation continues throughout the experiment. This is what we call a steady state error as shown in Figure 4.4, where corrections to the control input u_z is presented. We see that the P correction dominates the total correction and the I correction have almost no impact on the total correction. In cases with

steady state errors described in Section 2.2.2 the solution to cope with this kind of error is to increase the Ki parameter in the PID controller, but in this scenario the Kp should also have been increased to decrease the steady state error. The gains for the PID controller are presented in Table 4.3.

Kp	25
Ki	0.5
Kd	80

Table 4.3: PID gains used in Section 4.2.1

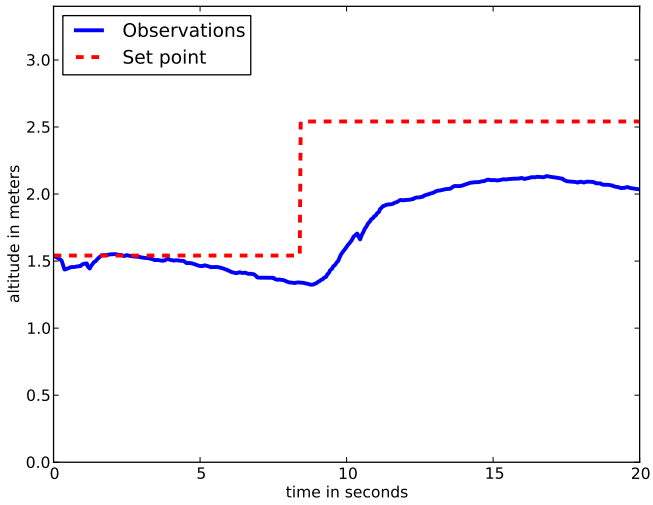

Figure 4.3: At the initial point is altitude of the drone is about 1.5 meters after about 8 seconds seconds set point was change to 2.5 meters. The set point is represented by the red line and the blue line represents the altitude. The closer the blue line is to the red dashed line, the more accurate the control of the drone is.

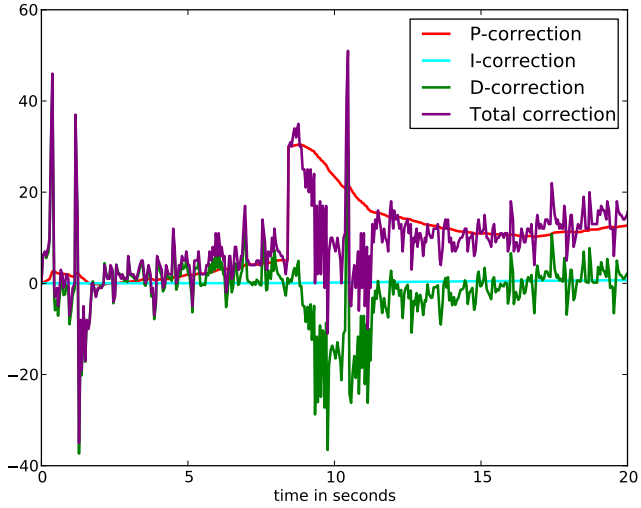

Figure 4.4: This figure shows the PID corretions issued to the control command u_z under the experiment. The maximum thrust was capped to 80 and the minimum to -50

Qualitative example

To present the qualitative performance of altitude hold an experiment where the some of the observations are subjected with heavy tailed sensor noise as described as outliers in Section 2.2.4. Figure 4.5 shows the outliers and the effect it has on the filter with and without the outier detection. The outlier detection handles the most heavy tailed outliers, but does not cache the cluster of outliers between 33 and 37 seconds. The threshold value for what is an outlier is could have been decreased to handle this. The threshold value in this experiment was 0.5.

Outlier detection

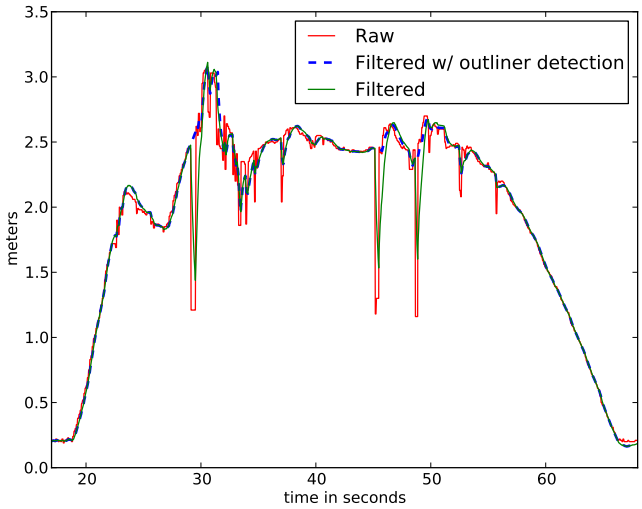

Figure 4.5: The figure shows the raw ultra sonic distance sensor measurements, filtered measurements and filtered measurements with outliers detection

4.2.2 Autonomous landing

The autonomous landing experiment was conducted as follows; The drone was manually flown to an altitude of 2.5 meters when the drone was stable and the marker was in the field of view of the camera the autonomous landing was enabled, as described in Section 3.7.7. The drone starts the decline after 9 seconds into the autonomous flight. Figure 4.6 and 4.7 shows the horizontal distance in roll and pitch direction from the landing spot through out the autonomous landing. The oscillation in roll direction is heavier than in pitch direction. From experiences made under this thesis the drone had an offset drift in both roll and pitch direction, but heavier to in the roll direction. This drift was most likely due to unbalanced weight distribution on the drone. This was experienced after changing batteries between tests. After the weight distribution was altered and the drones drift offset was altered as well. Figure 4.8 and 4.9 shows the corrections from the PID controllers applied to the control input to reach the set point. In Table 4.5 the PID gains used for this experiments are presented. To cope the oscillations we are seeing the Kd could have been set higher. After 10 seconds in

autonomous flight the drone starts the decline. The decline velocity set point was set to -0.20 m/s.Figure 4.11 shows the vertical velocity from the decline and had an average error from the set point at 0,06 m/s. The drone landed on marker with an error of 11 cm roll direction and 9 cm in pitch direction measured manually from the center of the drones body to the center for the marker. The drone was in autonomous flight for 25 seconds and decline took 16 seconds. Figure 4.14 shows snapshots from the video taken during this experiment. The video for this experiment can be found here; http://www.youtube.com/watch?v=9otkmpm6RUQ

	Roll	Pitch	Vertical velocity
Set point	0	0	-0.20 m/s
Average error	0.15 m	0.10 m	0.07 m/s
Variance	0.17 m	0.11 m	0.06 m/s
Standard derivation	0.11 m	0.06 m	0.06 m/s

Table 4.4: Statical results from the autonomous landing experiment. The results are error is logged from the PID controllers for the given control domain. The Roll and pitch controller errors is given in meters and the Vertical velocity in meters per second.

	Roll PID	Pitch PID	Autolanding PID
Kp	18	18	25
Ki	0.5	0.5	0
Kd	40	40	50

Table 4.5: The gains for the three PID controllers used for the autonomous landing experiment.

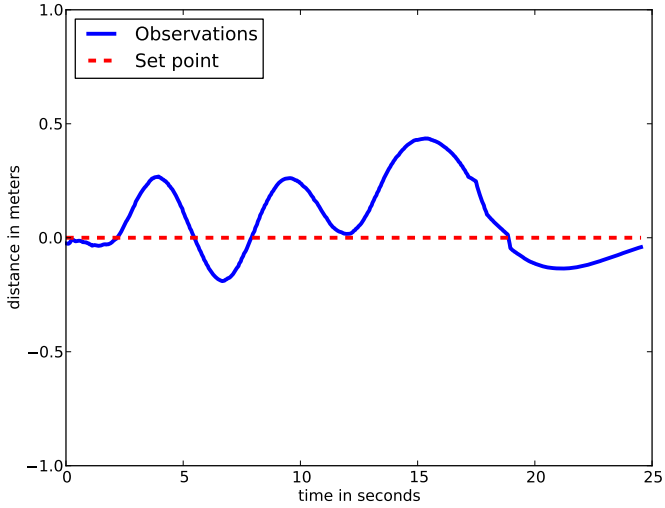

Figure 4.6: In this figure the distance in X direction from the marker is plotted. The blue line shows the estimated horizontal distance in meters as described in Section 3.7.3. The red dashed line the red dashed line is the PID controllers set point.

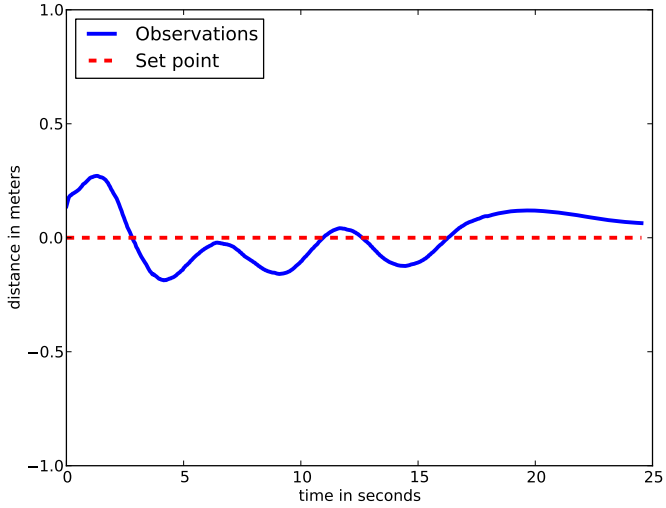

Figure 4.7: In this figure the distance in Y direction from the marker is plotted. The blue line shows the estimated horizontal distance in meters as described in Section 3.7.3. The red dashed line the red dashed line is the PID controllers set point.

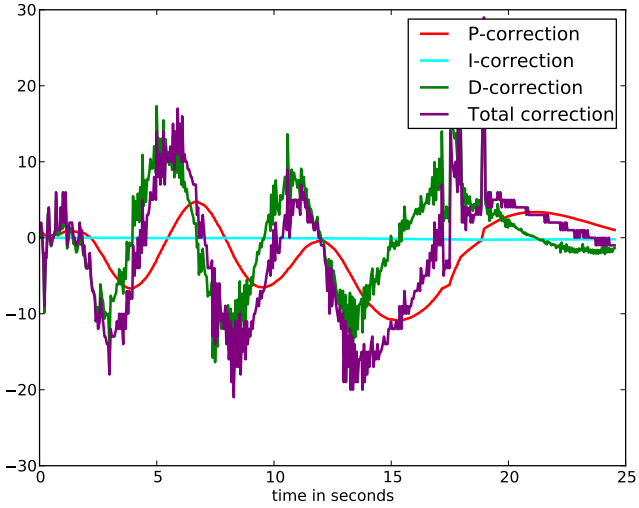

Figure 4.8: This figure shows the corrections to the control input u_Φ (roll) at the time t. The K_p gain was set to 18 , K_i gain to 0.1 and the K_d gain to 40. The maximum thrust was capped to 50 and the minimum to -50

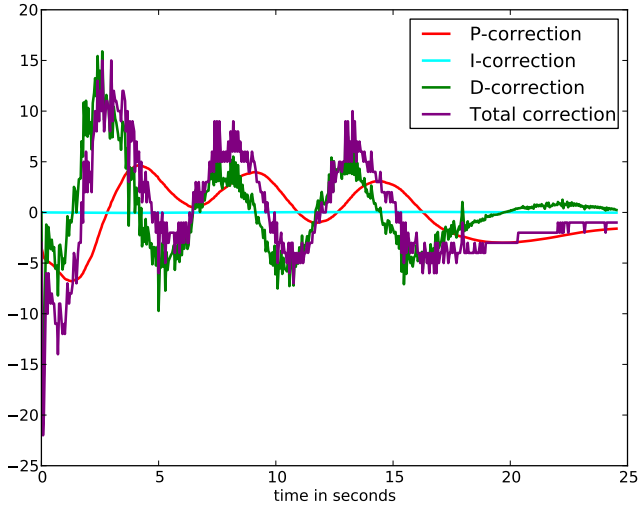

Figure 4.9: This figure shows the corrections applied to the control input u_Θ (pitch) at the time t. The Kp gain was set to 18 , Ki gain set to 0.1 and the Kd gain to 40. The maximum thrust was capped to 50 and the minimum to -50

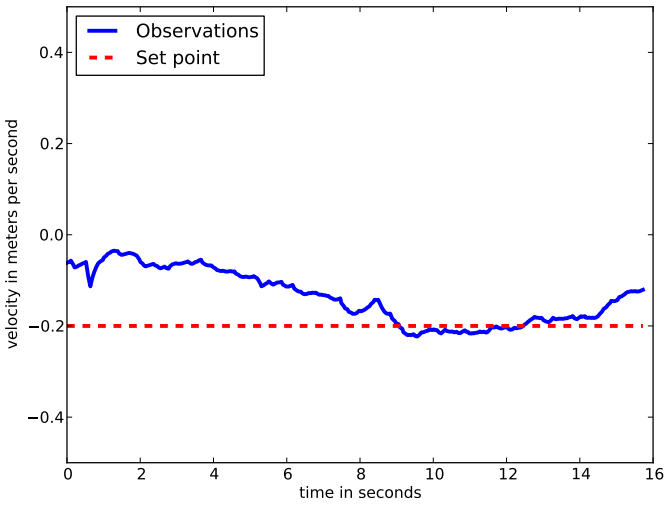

Figure 4.10: This figure shows the decline velocity during autonomous landing. The blue line represents the velocity and the red dashed graph is the set point for its PID controller.

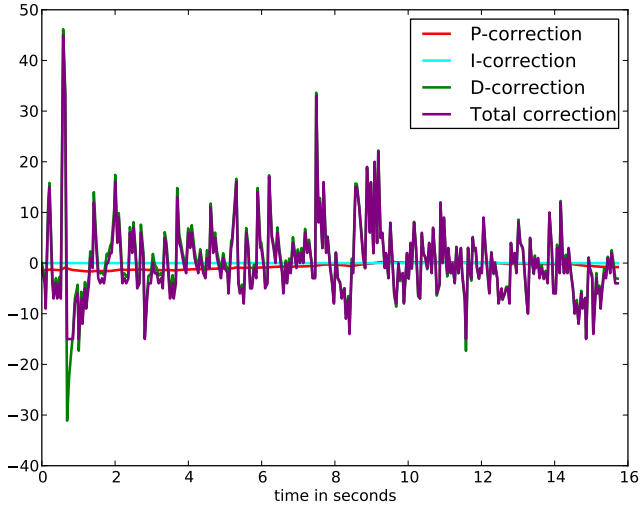

Figure 4.11: This figure shows the corrections applied to the control input u_z at the time t. The K_P gain was set to 25 , K_I gain to 0 and the K_D gain to 50. The maximum thrust was capped to 50 and the minimum to -15

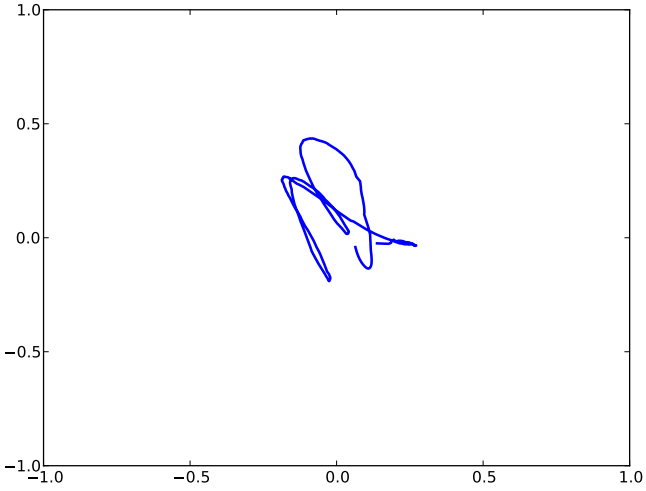

Figure 4.12: The graph shows the trajectory of the drone during an autonomous landing. The plot starts when the autonomous flight is enabled.

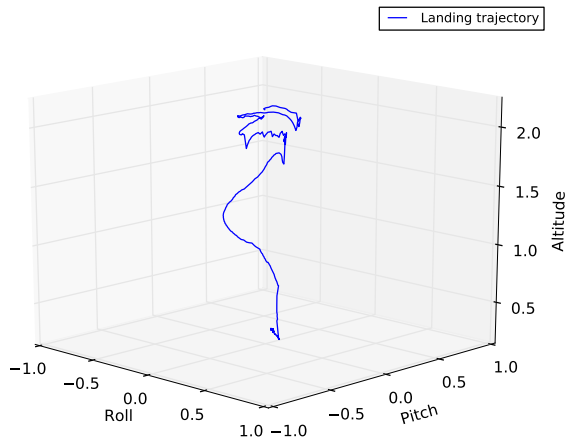

Figure 4.13: The graph shows the trajectory of the drone during an autonomous landing. The plot starts when the autonomous flight is enabled.

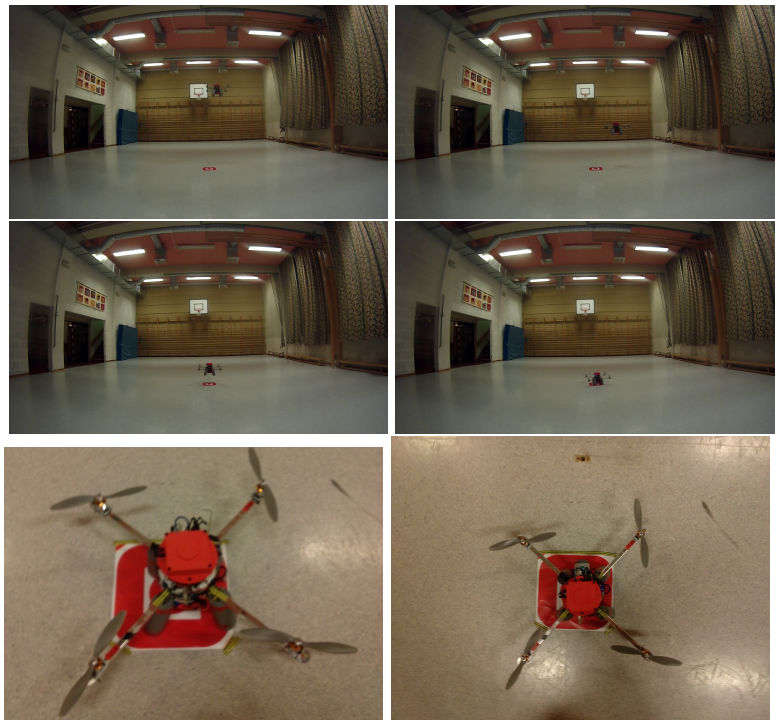

Figure 4.14: In the pictures above we see the drone performing an autonomous landing.

Quantitative example

The autonomous landing was tested a number of times through out this thesis. The tests results presented in Table 4.6 is the results from experiments where the drone was stable enough to give us real data. The assumptions for this was that the drone should operate without any significant drift in any direction due to unbalanced weight distribution. This unbalance originated from psychical parts on the drone that changes position throughout experiments. This was typically battery packs and parts remounted due to defection. In Table (4.6) the results from 7 stable experiments that was conducted are presented. The distance was measured manually with the shortest distance from the markers center to the drones body center.

Average Error	Standard derivation
0.27 meters	0.14 meters

Table 4.6: This table contains the results from 7 autonomous landings.

Qualitative example

For an qualitative example we have chosen to focus on robustness of the autonomous landing and the drones ability to recover if the marker observations are missing. Figure 4.15 shows the marker observations that represents the drones trajectory in X and Y direction. The red scatter points represents when the marker was out of the cameras field-of-view. Figure 4.16 shows the marker as its goes out of the cameras field-of-view and later makes it recovery based on the predicted position given by the Kalman filter that was applied.

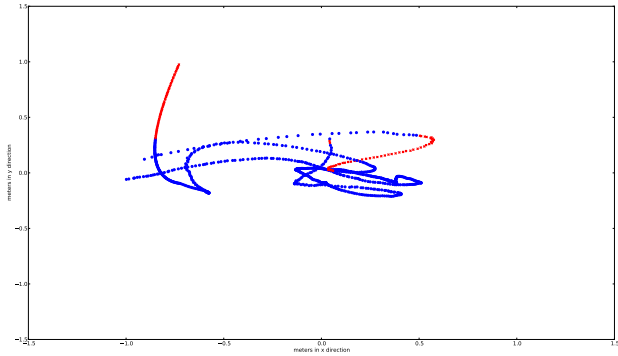

Figure 4.15: The blue scatter represents the estimated position of the landing spot when the marker with marker observations the red scatter represents the estimated position without marker positions, when the marker was out of the field-of-view

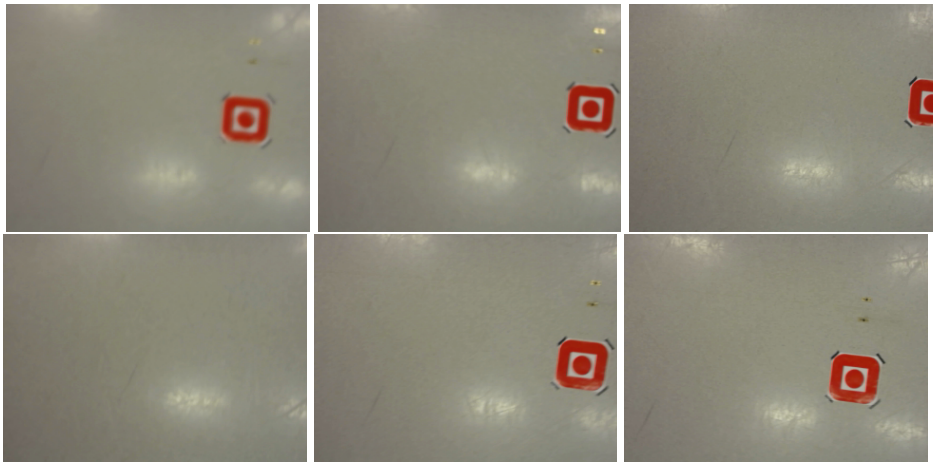

Figure 4.16: The figure shows video images taken from the drones camera. The marker goes out of the cameras field-of-view in picture 3 and 4, and making its recovery in image 5 and 6.

Chapter 5

Evaluation and Conclusion

In this thesis we have built a fully operating autonomous drone with with visual navigation abilities and all computations are done on-board with limited processing power. The drone was been built using only with off-the-self hardware a and with a budget under a $1000 3.2.5. We have developed software implementing computer vision algorithms for identifying the marker and calculating its position the linear Kalman filter for robust tracking of the marker and recovery after lost visual tracking. The Kalman filter implemented also fuses and filters noisy sensor observations. All the software developed [1] and used are open source, as well as the design for the 3D parts. [2]

A PID controller was implemented for control of the drone pose and fly to and hold desired set points. The Section 5.1 we discuss the performance of these implementations.

5.1 Evaluation

In Section 4.2.1 and 4.2.1 we showed the drones ability to hold an altitude over time and ability fly to a desired altitude. This is a important feature and affects the performance of the position hold and autonomous landing. In Section 4.2.2 we showed that drones ability the perform position hold and precise autonomous landing on a given landing spot. In Section 4.2.2 we have showed the robustness of the visual tracking and the drones ability to recover to position hold after a series of missing visual observations.

[1] `https://github.com/paalmoest/drone`
[2] `https://github.com/paalmoest/3d-drone`

5.2 Conclusion

We have in our experiments showed that the drone we have designed with our hardware and software implementation can perform under optimal conditions and in a indoor environment an autonomous precision landing within an average of 0.27 meters of the desired landing spot.

5.3 Discussion

On one hand, we can say that the system we designed is robust and it can recover from loss of visual tracking and handle noisy sensors and heavy outliers. But on the other hand this robustness is under what we call optimal conditions. The assumptions of these optimal conditions are that the drone initially operates should operate without any significant drift in any direction when no control command is issued. During this thesis this was not always the case and to fill the assumptions of the optimal conditions, manually calibrations of the weight distribution and tuning of the initial control commands values had be issued before any of the experiments. These calibration slashed our flight time in the experiments. The flight time for the drone at the end of thesis was about 6 minutes due to degraded battery capacity and the payload on the drone. In this thesis it has been difficult to distinguish between what is plain hardware issues EG. sensors giving over 50% outliers) and issues that could have been solved with a more robust and adaptive software implementations, like noise sensor data and attitude pose

The lack of outdoor experiments conducted is due to limitations in time and the weather conditions in Norway in December and January when the majority of the experiments was conducted. In Section 5.5 we have further discussed the limitations of the drone and suggested solutions.

5.4 Contributions

In this thesis we have showed the necessary steps to transform a radio controlled helicopter into a an autonomous drone. We have documented the what AI-methods need to be implemented to make it operational and

What are the main contributions made to the field and how significant are these contribution.

5.5 Future Work

5.5.1 Operating in challenging environments

In this thesis autonomous landing was only tested in a indoor environment. With landing outdoors more factors needs to be accounted for. Landing in windy environments may require a more robust and adaptive control of the drone. The identifying and tracking of the landing spot is done by color and shape (3.6.2) from experience made under this thesis the tuning parameters for marker recognition are directly influenced by the lighting condition in environment and search or learning algorithm could be applied to handle this challenges. The unscented Kalman filter could have been applied to leading to more robust and accurate results (Julier and Uhlmann. [1997]) of the state prediction.

5.5.2 Gimbal mount for camera

In this thesis the camera was mounted in a fixed position with a gimbal mounting would have enabled the camera to move in two direction and had been able to identify the marker from a greater distance. This would naturally have risen the complexity of the solution.

Bibliography

Aeroquad (2013). Aeroquad: The open source quadcopter / multicopter, http://www.aeroquad.com.

Anderson, B. D. O. and Moore, J. B. (2005). Optimal filtering. *New York: Dover*.

Arduino.cc (2013). Arduino mega 2560, http://arduino.cc/en/Main/arduinoBoardMega2560.

Bradsky, G. (2013). Open source computer vision library, http://docs.opencv.org/.

Engel, J. J. (2011). Autonomous camera-based navigation of a quadrocopter.

Faragher, R. (2012). Understanding the basis of the kalman filter via a simple and intuitive derivation. *IEEE SIGNAL PROCESSING MAGAZINE*.

Gross, B. (2013). Maxsonar operation on a multi-copter http://www.maxbotix.com/articles/067.htm.

Gstreamer (2014). Open source multimedia framework, http://gstreamer.freedesktop.org/.

Julier, S. and Uhlmann., J. (1997). A new extension of the kalman filter to nonlinear sys- tems. in proc. of the international symposium on aerospace/defense sensing. *Simulation and Controls*.

K, A. R. (2014). Contours - 2 : Brotherhood, http://opencvpython.blogspot.no/2012/06/contours-2-brotherhood.html.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *J. Basic Eng.*

Moore, D. S. and McCabe, G. P. (1999). Introduction to the practice of statistics. *WH Freeman and*.

Møst, P. M. (2013). Software repository for the drones software `https://github.com/paalmoest/drone`.

pykalman (2014). Pykalman: Kalman filter, smoother, and em algorithm for python, `http://pykalman.github.io/`.

Ryzhyk, L. (2006). The arm architecture.

Wescott, T. (2000). Pid without a phd. *Embedded Systems Programming*.

Appendices