



CHALMERS
UNIVERSITY OF TECHNOLOGY



Third-person Immersion

Vision-based Autonomous Target Tracking for Unmanned Aerial Vehicles

Bachelor's thesis in Computer Science and Engineering

Anton Lindgren
David Göransson
Emanuel Snellman
Jacob Hagstedt P Suorra
Daniel Larsson
Joacim Andersson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2014

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.

The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Third-person Immersion

Vision-based Autonomous Target Tracking for Unmanned Aerial Vehicles

ANTON LINDGREN

DAVID GÖRANSSON

EMANUEL SNELLEMAN

JACOB HAGSTEDT P SUORRA

DANIEL LARSSON

JOACIM ANDERSSON

© ANTON LINDGREN, June 2014.

© DAVID GÖRANSSON, June 2014.

© EMANUEL SNELLEMAN, June 2014.

© JACOB HAGSTEDT P SUORRA, June 2014.

© DANIEL LARSSON, June 2014.

© JOACIM ANDERSSON, June 2014.

Examiner: ARNE LINDE

Chalmers University of Technology

University of Gothenburg

Department of Computer Science and Engineering

SE-412 96 Göteborg

Sweden

Telephone + 46 (0)31-772 1000

Abstract

In recent years, Unmanned Aerial Vehicles (UAVs) have been entering new application areas. The objective of this thesis is to implement autonomous target tracking for a quadrotor UAV through the use of vision-based technology. The purpose is to implement a third-person view experience, where the UAV hovers behind and follows a targeted user, while sending a live video feed to a pair of virtual reality glasses, called Oculus Rift, worn by this user. Using vision-based tracking, one can minimise the overall equipment mounted on UAVs by utilizing the cameras that are already critical for the application area. Target tracking is achieved through the use of an image processing algorithm using Hough Circle Detection. A comprehensive analysis of this algorithm, also utilizing two different colour space filtering algorithms, is presented. Hardware choices made in building the quadcopter are also presented, supported by vibration test data showing the flight stability of it.

This thesis proposes a candidate model for using autonomous UAVs, with image processing as main navigational tool, in a third-person view application. Due to noise in the transmission of video sent from the UAV to the ground station that transforms image data to maneuvering commands, target user recognition have not reached the preferred level of accuracy, which have negatively affected the general performance of the application. This issue is presented and discussed.

Image processing is concluded to be a valid alternative as a way of implementing autonomous control of UAVs. Successful video transmission from the UAV to the Oculus Rift is demonstrated, as well as semi-successful autonomous control of the UAV.

Sammandrag

Under de senaste åren har obemannade luftfarkoster börjat tillämpas inom nya tekniska områden. Syftet med denna rapport är att presentera autonom målföljning för en obemannad farkost, med hjälp av bildanalysering. Målet är att implementera en tredjepersonsvy, där en quadcopter håller sig bakom och följer efter ett mål (användaren), samtidigt som den strömmar video till en Oculus Rift, ett par glasögon framtagna för användning inom virtuell verklighet. Med hjälp av bildanalysering kan man minimera den totala utrustningen på quadcoptern genom att utnyttja de kameror som redan är kritiska för applikationsområdet. Målsökningen uppnås genom en bildanalyseringsalgoritm vid namn Hough Circle Detection. En omfattande analys av denna, samt två olika färgfilteringsalgoritmer, presenteras. Även val av den hårdvara som behövs i ihopsättningen av en quadcopter presenteras och stöds med data från vibrationstester som visar flygstabiliteten.

Denna rapport föreslår en modell för att autonomt styra luftfarkoster med bildbehandling som huvudsakligt navigeringsverktyg, i en tredjepersonsvy-applikation. På grund av bildstörningar vid videoöverföringen från quadcoptern till markstationen som översätter bilddata till styrkommandon, har måligenkänningen inte nått en tillräcklig precision. Detta har negativt påverkat systemets prestanda. Detta problem presenteras och diskuteras i rapporten. Bildanalys som ett sätt att implementera autonom flygning utvärderas och en positiv slutsats dras. Lyckad bildöverföring från quadcopter till Oculus Rift presenteras, tillsammans med delvis lyckade försök till autonom styrning av quadcoptern.

Vocabulary

| | |
|--------------------------|---|
| APM | Ardupilot Mega, an inertial measurement unit autopilot based on the Arduino Mega platform. |
| ESC | Electronic speed controller, an electronic circuit used to vary the speed of a brushless electric motor. |
| FOV | Field of view. |
| Frame grabber | USB device for converting analog video to digital video to be used with a computer |
| HSV | Colour model, stands for Hue, Saturation and Value. Value is a separate light channel |
| IMU | Inertial measurement unit, an electronic device used to measure movement data from the vehicle. |
| Oculus Rift | A pair of head-mounted glasses with displays used to give the user a virtual reality. |
| PAL | Phase Alternating Line, a colour encoding system for analog video |
| RGB | Colour space containing Red, Green, and Blue. |
| Third-person view | A bird's-eye view of the person as viewed from behind the head. |
| YCrCb | Family of colour spaces. Containing a luma component as well as blue-difference and red-difference chroma components. |

Contents

| | |
|--|-----------|
| 1 Introduction | 8 |
| 1.1 Background | 8 |
| 1.2 Purpose | 9 |
| 1.3 Delimitations | 9 |
| 1.4 Prospective applications | 9 |
| 1.5 Method | 10 |
| 2 Hardware design choices | 11 |
| 2.1 Quadcopter | 12 |
| 2.2 Choice of autopilot | 13 |
| 2.3 Video resolution presented to the user | 13 |
| 2.4 Mobility and ease of use | 14 |
| 3 Software implementation | 15 |
| 3.1 Domain model | 15 |
| 3.1.1 Cam-Share | 16 |
| 3.1.2 Image Processing | 16 |
| 3.1.3 Oculus Distortion | 16 |
| 3.1.4 Quadcopter Control | 17 |
| 3.2 Software communication protocols | 17 |
| 3.2.1 Pipes | 18 |
| 3.2.2 Shared memory | 18 |
| 3.3 Programming languages | 19 |
| 4 Quadcopter | 21 |
| 4.1 Components and assembly | 21 |
| 4.1.1 Autopilot | 22 |
| 4.1.2 Radio signal receivers | 22 |
| 4.1.3 Camera | 23 |
| 4.1.4 Gimbal mount | 23 |
| 4.1.5 Video transmission | 23 |
| 4.2 Aerial maneuvering | 24 |
| 4.2.1 Roll | 25 |
| 4.2.2 Pitch | 25 |
| 4.2.3 Throttle | 25 |
| 4.2.4 Yaw | 25 |
| 4.2.5 Flight modes | 26 |
| 4.3 Automatically maneuvering the quadcopter | 26 |
| 5 Imaging | 28 |
| 5.1 Oculus Distortion | 28 |
| 5.2 Image Processing | 29 |
| 5.2.1 OpenCV | 32 |
| 5.2.2 Target tracking | 32 |

| | |
|---|-----------|
| 5.2.3 Priority system | 35 |
| 5.2.4 Debugging | 37 |
| 6 Testing | 38 |
| 6.1 Quadcopter speed | 38 |
| 6.2 Video latency | 38 |
| 6.2.1 Test data | 40 |
| 6.3 Vibration | 41 |
| 7 Results | 42 |
| 7.1 In-flight video transmissions | 42 |
| 7.1.1 Immersive user experience | 43 |
| 7.2 Attempting autonomous control through image processing | 43 |
| 7.3 Image processing evaluation | 44 |
| 8 Discussion | 45 |
| 8.1 Negative aspects of using image processing for autonomous control of UAVs | 45 |
| 8.1.1 Video latency analysis | 45 |
| 8.1.2 Image noise analysis | 46 |
| 8.2 Positive aspects of using image processing for autonomous control of UAVs | 47 |
| 8.2.1 Predicting movements using previous data | 47 |
| 8.3 Implementing a third-person view in real life | 48 |
| 8.4 Quadcopter speed and vibration tests | 48 |
| 8.5 Autopilot board damage | 49 |
| 8.6 Future development potential | 50 |
| 8.6.1 Real-world applications | 52 |
| 8.7 Ethics | 52 |
| 9 Conclusions | 53 |
| References | 54 |

1 Introduction

1.1 Background

During the past few years, the use of Unmanned Aerial Vehicles (UAVs) has grown rapidly within various research areas as well as civilian and military applications. There are a few different types of UAVs, all with different flight dynamics. One of these is the quadcopter, which is propelled by four rotors, and manoeuvres by altering the rotation of the four rotors in various ways. With its unique design and small size it is very agile and has the ability to loiter and perform complex aerial manoeuvres. Omnidirectional flight capability also extends its flight dynamics and movability. Because of its manoeuvrability, quadcopters are planned to be used in civilian applications such as firefighting (Hatchet, 2014) and rescue operations (Solon, 2013).

In most applications that provide autonomous control of UAVs, non-visual sensors such as gyros, accelerometers, inertial measurement units (IMU) and global positioning systems (GPS) are utilised in order to position the vehicle properly (Azrad, 2010, p.255). This can limit the amount of application-critical equipment that can be mounted to the quadcopter, due to the fact that it needs four motors to operate and therefore consumes more power than traditional UAVs. In order to get around this problem, researchers are analysing the viability of image data as main positioning tool. Using a vision-based technique comes with a few advantages other than decreasing the overall size and weight of the quadcopter. One of these is the decoupling from other critical systems, such as GPS. As a result of this, vision-based UAVs can operate in environments that traditional UAVs can not.

The work presented in this thesis addresses the problem of autonomous target tracking through the use of vision-based technology.

1.2 Purpose

This thesis describes the planning, design, and implementation of a system in which a quadcopter autonomously follows and films its target at close range, with the intent of creating a third-person view experience. The quadcopter is to maintain a constant distance from its target, while adapting position and rotation in order to face the camera towards the target.

Image processing will be used as the main navigation tool in the system. As this is not a widely used technique for navigating UAVs, this puts emphasis on the theoretical nature of this thesis.

For clarity, the thesis is divided into two separate tracks: evaluating whether or not image processing is a good way of implementing autonomous control of airborne vehicles; and assessing the quality of the immersive experience achieved by the third-person view.

1.3 Delimitations

Since the development of a quadcopter is not among the purposes of this project, commercial products will be used in assembling it. This goes for both mechanical parts, such as the frame and the motors, and for the software used to interpret radio signals into electromagnetic pulses.

Due to time limitations, the quadcopter will not be fully aware of its surrounding in the sense that it can steer away from all objects in its path. As an implication of this, testing will only be done in open spaces where the problem does not arise. The project does not focus on developing image analysis software, but merely utilises available technologies in order to meet the objective.

1.4 Prospective applications

The results of this thesis could be seen as a candidate model for autonomous control in applications in which a camera is mounted to a quadcopter. Utilizing the equipment already mounted to it, one can minimise its weight and power consumption. This can be useful in various situations where the need of a bird's-eye view is required, including riots, firefighting and on various applications within the army. Not having to control the quadcopter by hand gives the user freedom to meanwhile do other things.

Another prospective application is within the field of entertainment, perhaps gaming in particular. Games with a digital third-person view are common, and this thesis will bring a correct, physical third-person view into the equation.

1.5 Method

Due to the small size of the project team and the fast paced development that was required, agile methods has been used throughout the project. A lightweight version of Scrum¹ has essentially meant fast prototyping through the use of existing frameworks, volatile and regulated requirements, and adaptable groups and roles within the team. During weekly team meetings, the previous week's priorities has been reviewed and adjusted. The workload and team roles for next week were set, and this iterative process carried on throughout the project, making regular adaption to changing circumstances and an unexpected level of progress easy.

Throughout the project, an approach with iterations of researching, implementing, and finally testing has been used. This has been effective in both assessing what solutions are viable, as well as evaluating the solutions that were implemented. This approach has been effective throughout the project, since it has made research faster and the results of it easier to comprehend.

Software has mainly been developed in groups of two. Emphasis has been put on the importance of co-operative development, and the weekly meetings has brought up any major issues that team members have. These issues has then been considered in the following week's priorities, often meaning that one or several of the other team members assist in solving the problem at hand. Research that has been carried out is evaluated to determine if the researched technology is a viable part of the solution.

¹ For more information about scrum see (Scrum Alliance, n.d.)

² For more information about the example application the Oculus Distortion module is based on see (remote-eyes -

2 Hardware design choices

This chapter will highlight some of the choices made in regards to what hardware to order and assemble. The issues and choices presented below represent the ones that are considered to have influenced the project the most. Some of these choices largely impact the details for a couple of the quadcopter components. As a result of this, design choices are presented and discussed in this chapter before Chapter 4, which will go into detail on the quadcopter components that were chosen for this project.

2.1 Quadcopter

High customizability as well as precise control were two priorities when considering quadcopter solutions. This has meant that most of the research focused on comparing different open source software solutions and their supported hardware platforms rather than closed end-user products.

This resulted in more time being spent assembling and configuring the quadcopter as well as grasping how the open source software should be used together with its hardware .

The final quadcopter solution consists of the components listed in Table 1.

Table 1. Quadcopter hardware parts

| | |
|--------------------------------|---|
| IMU AutoPilot | ArduPilot Mega (APM 2.6) |
| Frame | Flamewheel F450 |
| Landing Gear | DJI F450 Flamewheel Landing Gear |
| Motors | 4x DJI 2212 Brushless 920 kV motor |
| Speed Controllers | 4x DJI 30A Opto ESC |
| Height Sensor | Maxbotix HRLV-EZ4 |
| Radio Set | 3DR Radio Set 433 Mhz |
| Remote controller and reciever | WFly WFT07 2.4 GHz 7-kanaler, incl receiver |
| Gimbal | DJI Phantom Brushless Gimbal w/ Simple BGC |
| Camera | GoPro Hero 3 White |
| Video Link | AH-5.8 GHz 200 mW |
| Battery | Gravity 5500 mAh 3S 30-40 C LiPo |
| Charger | EV-Peak V6AC 50 W Multi charger 230 V/12 V |

2.2 Choice of autopilot

A quadcopter needs an autopilot to be able to keep stable as it is inherently instable (Niroumand, F-J., 2013). For this project, the IMU Autopilot APM (Ardupilot Mega) was used. The APM is based on an open-source platform called Arduino Mega and is capable of autonomous stabilisation, two-way communications with telemetry, and navigation.

The APM consists of specialised hardware and well documented software. This, combined with the fact that the software is open source, motivated the choice of using it for the project.

More about autopilots in chapter 4.1.

2.3 Video resolution presented to the user

In order to achieve an experience as realistic as possible, the user should not be able to distinguish pixels in the images displayed by the Oculus Rift. However, the system has two bottlenecks when it comes to resolution: the Oculus Rift and the hardware used for analog video transmission.

The developer version of the Oculus Rift that has been used during the project has a relatively low resolution at 1280×800 which results in 640×800 pixels per eye (Oculus VR, 2012). The low resolution has been criticised from the launch of the developer kit (Orland K., 2013) and is said to be improved for the consumer release. In fact, the second version of the developer kit, which was released after this thesis project had started, increases the resolution to 1920×1080 giving each eye 960×1080 pixels (Oculus VR, 2014). With these developments in the Oculus Rift's resolution, this should not pose as an issue had this project been continued on to develop a more thorough solution.

Furthermore, the analog video transmitter that streams the video from the quadcopter to the ground station only supports a resolution of 640×480 pixels. The difference between analog and digital transmission from the GoPro can be seen in Figure 1.

Analog transmission was chosen over digital transmission because research done early in the project favored this. This was partly because the GoPro camera that was decided to be used has a long latency for its built in WiFi video transmission (martcerv, 2010). Analog transmission was also recommended by multi-rotor expert Mr M Hjalmarsson, when interviewed on 29 Jan 2014.

Thus, the limit for end-user resolution is set by the video link used to transmit the video from the quadcopter to the ground.

Other cameras aside from the GoPro Hero 3 could have been used. There exists so called FPV cameras made specifically for RC vehicles that most likely could fit the needs of the project. Meaning they are lightweight, small and can transmit their video well. One of the bigger reasons for choosing the GoPro was that a gimbal was to be used and many of these are made specifically for the GoPro. The GoPro is an extreme sport camera made to withstand vibrations and deliver high quality images and it is also lightweight and small enough therefore the GoPro was considered a good choice of camera.

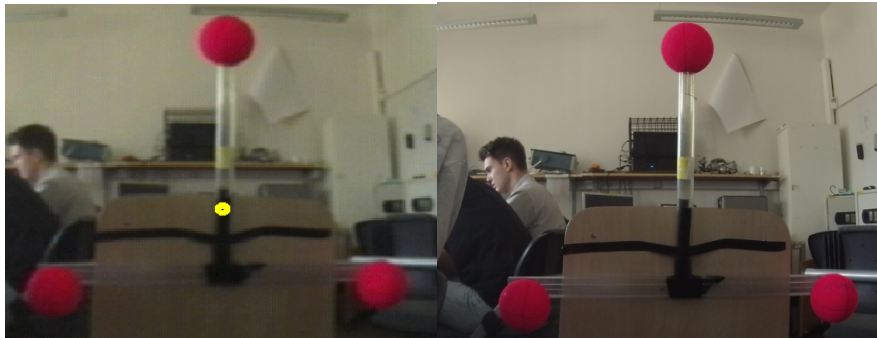


Figure 1. Analog quality (left) and digital quality (right) from GoPro Hero 3 White

2.4 Mobility and ease of use

The ideal system would be easily transportable by the user. When ordering and assembling the hardware, focus has been put on documentation transparency and customisation more than mobility and durability. Therefore, the prototype is quite cumbersome and requires much knowledge of both the hardware and software before it can be used.

This cumbersomeness is a by-product of choosing open source components rather than closed end-user commercial products, which are more geared towards, and designed for, mobility and durability. This downside is however conquered by the fact that the project team has been able to maintain and troubleshoot single components within the system when issues have arisen.

3 Software implementation

Chapter 2 explains the hardware design choices that is believed to have influenced the outcome of the project the most. This chapter will explain and motivate the way the software was implemented.

In the beginning of the project, a choice was made to focus on modularising the system's software components as much as possible.

These subchapters describes how the modules were derived from the domain, as well as briefly specifies the techniques used to achieve the modularity needed.

3.1 Domain model

The software is created in a module-based design and consists of four core modules, which together provides the functionality of the concept (see Figure 2).

Below, each module will briefly be described as for purpose, and when appropriate, a short paragraph on the approach taken.

Theory and important implementation details are presented Chapter 4 and Chapter 5.

The whole domain relies on receiving a video stream which is utilised in two different ways. Firstly, it is output to screen, to be displayed on the Oculus Rift. Secondly, it is used as input to control the quadcopter. The video stream is read by the Cam-Share module from a camera, which distributes it to the Oculus Distortion and Image Processing modules. Oculus Distortion is an end-point for the video as it displays the stream on screen, whereas Image Processing is an intermediary step, and outputs a 3D position utilised by the Quadcopter Control module.

For repositories and basic usage documentation of the software please refer to Appendix A. All the modules inside this core package have been implemented for this project, by the project group, except when stated otherwise.

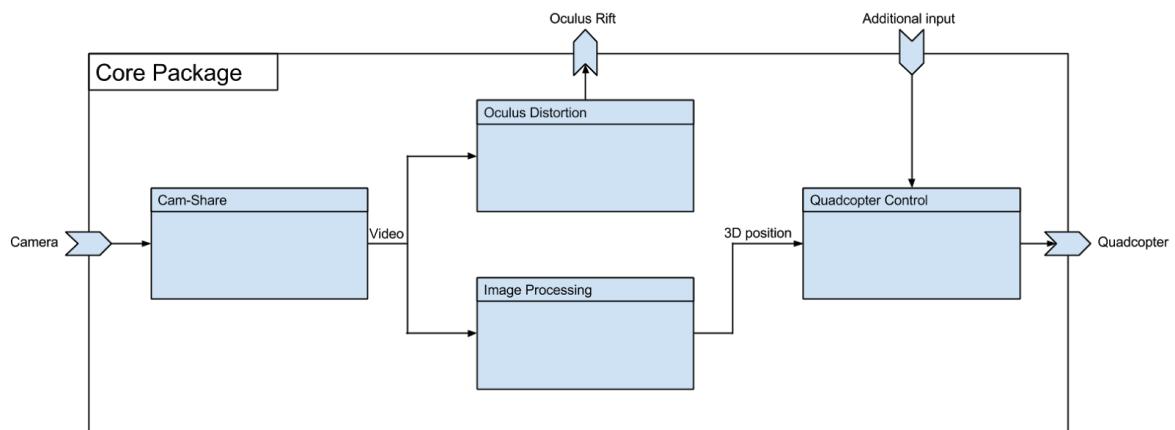


Figure 2. Domain model

3.1.1 Cam-Share

The system requires two independent copies of the video stream from the camera. This is solved by writing the latest frame to a shared memory segment that other processes can access. This enables multiple processes to simultaneously access the video stream, thus enabling the possibility of running Oculus Distortion and Image Processing as two separate processes.

3.1.2 Image Processing

The image received from the Cam-Share module is copied and analysed. The output of the Image Processing module is the position data, consisting of a 3D vector and angles describing the relations between the camera and the user.

The position data is obtained by equipping three preset objects on the target and track these. To be able to calculate a relative position from the camera to the user these objects are triangulated so that pixel dimensions can be compared to real world measurements.

3.1.3 Oculus Distortion

For the user to be able to see the image properly in the Oculus Rift, the video output from the Cam-Share module must be transformed into a format appropriate for the Oculus Rift.

To achieve this, a modified example application² from Oculus VR is forked.

² For more information about the example application the Oculus Distortion module is based on see (remote-eyes -

3.1.4 Quadcopter Control

The Quadcopter Control module utilises input from other modules to determine commands to the quadcopter. These commands are sent to the quadcopter through a radiolink making the quadcopter follow its target. The Quadcopter Control module only handles data from image processing, but the module-based design of the system enables further development of this module to add additional modules that help to improve the regulation of the quadcopter.

3.2 Software communication protocols

It was apparent from the start of the project that the software developed would become quite complex over time. Each module in the design, which was described in the Domain Model chapter, would therefore need to specify its exact output. This resulted in a system where each module compiles its own binary, and all communication is handled by the operating system with input- and output streams through *pipes*³ when possible. When pipes are inadequate for a communication channel, *Shared Memory*⁴ is used. This architecture is inspired by the UNIX principle of doing one thing and only that (Gancarz, 2003).

Having each module run as a standalone binary enabled the system to include modules written in different programming languages. This allows for implementing computationally heavy and performance critical applications in performant programming languages such as C++, and modules that are not performance critical in languages that are easy to prototype in, such as Python.

Videograbber to Oculus Rift for FPV - Google Project Hosting, n.d.)

³ For more information about pipes see (pipe(7) - Linux manual page, 2005)

⁴ For more information about shared memory see (shm_overview(7) - Linux manual page, 2010)

3.2.1 Pipes

When possible, modules use the standard output from a program, called `stdout`, to communicate their output. Further, all input variables are also read from the standard input, `stdin`, when possible. This makes it easier to combine the modules within the system, and has the advantage that they can be written in different languages, as well as completely without knowing the internal structure of the other modules.

```
$ modA | modB > logfile
```

Figure 3. Piping from module A to B, then piping from B to a logfile

This method can not be applied as easily when a module must be able to handle writing output to several other modules, as implied in Figure 3 the syntax has no natural way of splitting the output through a pipe. Following the same reasoning, reading input from several pipes would require a less standardised approach, which decreases the value for pipes in this solution.

3.2.2 Shared memory

When pipes are not applicable, modules use shared memory for communication. This method is especially useful when several processes depend on a stream of data produced by the same I/O device. In this case, that is the camera.

The contrary to shared memory would be for one module, the owner module, to take ownership of the I/O device and then produce output for one or several other modules, the reading modules. This solution would make the performance of the reading modules dependent on the performance of the owner module. For instance, if the owner module becomes slow at some point, the reading modules would suffer from low frame rate, in the case of a camera as a source.

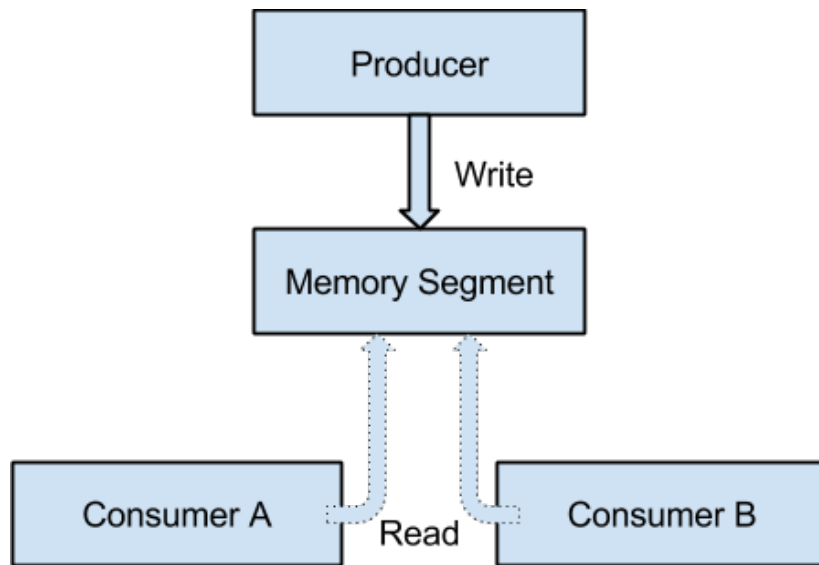


Figure 4. A simple producer-consumer visualisation

The shared memory implementation for this project uses one producer and two consumer modules (see Figure 4). This means that all consumers depend on the producer to exist and perform well (maximise frame rate at a low computation cost). This setup requires only a minimal producer module, which in this case only captures the frame and writes it to the shared memory segment. Having this minimal producer reduces the risk of a bottleneck occurring at the camera capture.

The implementation of the producer-consumer pattern is simplified in the sense that it does not ensure mutual exclusion on the memory segment, and thus does not require a queue to put elements in. This has the drawback that a consumer can read parts of an old frame and parts of a new frame, as well as read the same frame multiple times. However, this has not had any noticeable or measurable effects in the regulation of the quadcopter, nor has it produced any visual distortion for the user. Therefore, solving this issue has not been prioritised.

3.3 Programming languages

The programming language Python was used in the development of the quadcopter control module. The main reason for this is that the quadcopter control module was built on top of the existing code base for MAVProxy, which was coded in Python.

The OpenCV library used for the image-processing software could be used with a variety of programming languages, such as C, C++, Python, Java, and Matlab (OpenCV, 2014). For the image analysis parts of this project, performance was a major point in deciding which

programming language to use. Another thing taken into consideration was how good the documentation was for the different languages, where C and C++ turned out to have a much greater collection of documentation and examples than Java or Python. Therefore, C++ was the desired language for this part of the project. There is a possibility that coding it in C would have provided a small performance boost, but during runtime, it was found not to be necessary.

The Cam-Share and Oculus distortion software also uses C++ as the developing language. The main reason for why Cam-Share uses C++ is because it was ported to handle the camera input from the already written parts within the Image Processing module. This made it easy to port the capture part into a new program. The oculus distortion program was, as mentioned when introducing the module, a fork of an already finished program which was written in C++. Also, the Oculus Rift examples provided by the Oculus Rift SDK were mostly written in C++.

4 Quadcopter

This chapter will go into detail about the theoretical concepts that are important for understanding how the system works and why certain decisions were taken. Mostly, this will be about describing aerial concepts for the quadcopter and how the Quadcopter Control software module is utilised for autonomous control.

4.1 Components and assembly

In Chapter 2 the full list of quadcopter hardware in the system is specified along with some decision highlights. This chapter will describe, in detail, the individual components found on the quadcopter. For visualisations on the different parts, please refer to Appendix C I.

The quadcopter used in this project has four propellers which are assembled on a cross-shaped rigid frame. The propellers are installed as counter-rotating pairs, one pair that spins clockwise and another that spins counter-clockwise (see Figure 6). This setup removes the need for a tail rotor and prevents the quadcopter from spinning around its vertical axis (Bresciani, 2008).

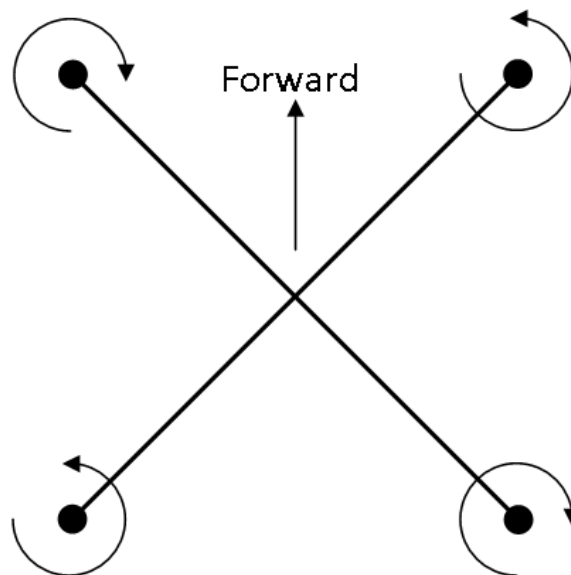


Figure 6. Motor layout as seen from above

The propellers are mounted on four motors with each motor being controlled by an Electronic speed controller (ESC). An RC transmitter is installed on the quadcopter to enable a microcontroller or a computer to control the ESCs. A video transmitter is installed to enable video feed transfer from the camera to a computer.

To power the quadcopter, a three cell lithium polymer battery with a capacity of 5500 mAh and a discharge rating of 30C is used. The C rating is a measure of the rate of which the battery discharges relative to the maximum capacity of the battery. For this battery $5500 \text{ mAh} * 30\text{C} = 165000 \text{ mA}$ may be continuously drawn from the battery.

4.1.1 Autopilot

Autopilots are used together with quadcopters as they are inherently instable. By using sensors that measure the angular velocity, orientation, linear acceleration and other important data from the quadcopter, the autopilot can control the quadcopter to keep it stable. To measure data, an IMU is added. The IMU measures the acceleration with accelerometers and detects rotational changes with gyroscopes (Deyle, T.J. 2008). IMUs with inertial sensors are commonly used to improve flight stabilisation and autonomous hovering for helicopters or quadrotors (Höflinger et al, 2013).

The sensor data collected by the IMU is used by the autopilot which parses the data and sends commands to the motors and servos on a remote controlled vehicle.

To prevent vibration from disturbing the autopilot some cushioning is installed between the frame and the autopilot board.

4.1.2 Radio signal receivers

To be able to control the quadcopter from the ground, a radio link is needed. In this project, two different radio receivers are being used. One which handles radio signals from a microcontroller, used by a human for testing purposes, and one used to receive radio signals from the computer. The computer uses a 3DR Radio Set, communicating at the frequency 433 MHz. The 3DR Radio Set modules are interchangeable and uses two-way full-duplex communication which means the modules can both send and receive data simultaneously. Duplex communication is useful since the autopilot can send feedback data on the same radio link that the computer uses to send the control commands.

The radio link uses a protocol called MAVLink. MAVLink is a lightweight header-only message communication protocol. It is designed to be used with air vehicles to transmit messages with high speed and safety⁵.

⁵ For more information about MAVLink see (MAVLink Micro Air Vehicle Communication Protocol - QGroundControl GCS, n.d.)

4.1.3 Camera

A camera is needed to provide video for the Oculus Distortion and Image Processing modules explained in section 3.1. As discussed in section 2.1 it is important the camera has low latency, is lightweight and has a high resolution.

As an implication on how the quadcopter maneuvers, the quadcopter will tilt when moving on the horizontal plane. To avoid this affecting the video feed, a gimbal mount is used for stabilisation of the camera.

4.1.4 Gimbal mount

Gimbal mounts are a way of keeping the camera fixed on the horizontal plane, they are often custom made for specific camera models or brands.

The gimbal mount used in this project is a device with two mutually perpendicular and intersecting axes of rotation, this makes it possible for a camera attached to the gimbal mount to have free angular movement in two directions, in this case on the roll and pitch axes. The gimbal mount is equipped with two motors, one for the lateral and longitudinal axis each. These motors actively try to keep the camera in a stabilised position by counteracting the quadcopters angular movements.

4.1.5 Video transmission

Analog video transmission is used to send a video feed from the camera mounted on the quadcopter to the computer. For this, a video transmitter is mounted on the quadcopter. The video transmitter works on the frequency 5.8 GHz with a signal strength of 200 mW. According to the transmitter manual⁶, a signal strength of 200 mW has a working range of about 100 meters. Since the quadcopter is never supposed to be more than a few meters behind the user, a signal strength of 200 mW is sufficient.

On the ground, a video receiver connects to a frame grabber, which converts the analog video to digital, making it available for the computer. The image is sent to the Cam-Share software, which in turn can share it to both the Image Processing and Oculus Distortion modules.

⁶ See (BOSCAM TS 351 - RC 805 Manual. n.d.) for the manual

4.2 Aerial maneuvering

A quadcopter is able to rotate in three dimensions, commonly referred to as roll, pitch, and yaw. Roll is rotation around the longitudinal axis, pitch is rotation around the lateral axis and yaw is rotation around the normal axis. This can be seen in Figure 5 where the longitudinal, lateral and normal axis is represented by Z, X and Y respectively. The quadcopter can do these rotations by adjusting the four rotors in certain ways. Quadcopters commonly expose separate radio channels for each of these dimensions through their autopilot, with a fourth channel for handling throttle.

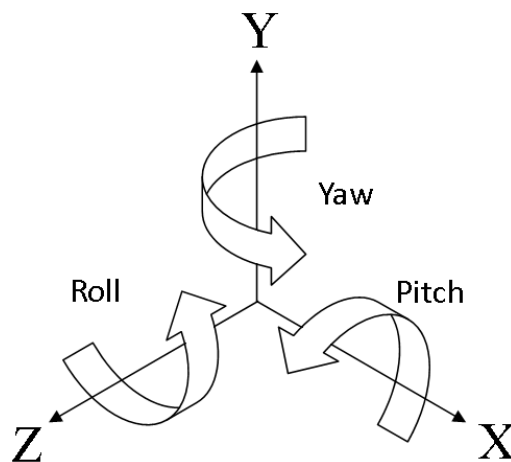


Figure 5. The three axes upon which quadcopters can maneuver

The following sub chapters will describe the four radio channels that were used in this project in order to control the quadcopter, both through the RC controller for testing, as well as through the Quadcopter Control module that was introduced in Chapter 3.

To complement the different radio control channels that are used to maneuver the quadcopter, the APM used as autopilot utilises different flight modes. Each of these flight modes slightly modifies the autopilots interpretation of input from the channels, often shifting between more or less autonomous control.

4.2.1 Roll

Manipulating roll has the effect of moving the quadcopter left or right (when no other channel is manipulated). Rolling clockwise is accomplished by decreasing the thrust on the two rotors on the right side, and increasing the thrust on the two rotors on the left side. Similarly, rolling counter clockwise is accomplished by decreasing the left side rotors' thrust and increasing the right side rotors' thrust. The combined vertical thrust remains the same, which means that the quadcopter will move left or right while maintaining the same altitude.

4.2.2 Pitch

Pitch is the action of tilting the quadcopter forwards or backwards to provide forward or backward momentum. To tilt the quadcopter forwards, thrust is increased on the two rotors at the back of the quadcopter, and the thrust is decreased on the front two rotors. This results in a forward tilt, while maintaining the total combined vertical thrust which causes the quadcopter to move forwards, while keeping the same altitude as before. Analogously, to move the quadcopter backwards, thrust is increased on the front pair rotors, and decreased on the back pair rotors.

4.2.3 Throttle

Throttle is used to increase or decrease the rotation of all four motors simultaneously. When increased, it produces a force downwards, which makes the quadcopter go up in a straight motion and gain altitude. When decreased, the force downward is reduced, which results in a loss of altitude. When used in conjunction with roll and/or pitch, different movements can be accomplished.

4.2.4 Yaw

The yaw channel is used to make the quadcopter rotate around the normal axis (commonly referred to as the Z or vertical axis). This is done by increasing the thrust on the clockwise pair rotors, and decreasing the thrust on the counter-clockwise pair rotors (or the other way around, for rotating in the opposite direction). The combined total vertical thrust remains the same, so the only movement that occurs is rotational movement around the normal axis.

4.2.5 Flight modes

The autopilot has several built in flight modes. The different flight modes range from simple stabilisation, still allowing manual control of all channels of the quadcopter, to fully automating the flight and navigating the quadcopter by GPS.

To achieve the movements required for this project a mode called altitude hold was used exclusively. Altitude hold mode is an automatic state where the autopilot controls the thrusts on the motors to ensure it keeps a stable and constant altitude while allowing yaw, pitch and roll to be controlled normally. The throttle channel is controlled autonomously as long as the controller has the throttle in its middle position. If the throttle is increased, the altitude at which the quadcopter will hover is increased, and if it decreases the altitude will decrease. This means that the quadcopter can be manoeuvred the desired way by controlling yaw, pitch and roll without having to worry about keeping the altitude.

4.3 Automatically maneuvering the quadcopter

As described in section 4.1.1 an autopilot will be used to stabilise the quadcopter, but for controlling the quadcopter a separate computer on the ground will be used. It will handle the image-processing calculations that controls how the quadcopter should move in order to stay behind its target, as well as transmit the actual data packets to the quadcopter radio receiver.

For the quadcopter to follow its target, software is needed to automatically control the quadcopter according to feedback about the quadcopters location. The automatic control is accomplished by first having the software command the quadcopter to go into altitude hold mode, and in that mode override the RC channels and transmit the commands over the radio link. The software has the ability to override the four primary RC channels: roll, pitch, throttle and yaw. The throttle channel is used to set the altitude at which the quadcopter will hover. When the altitude is set, the three other RC channels are manipulated to move the quadcopter towards the correct position.

To achieve this, a module has been developed to be used with an existing code base called MAVProxy⁷. MAVProxy is a command-line-based ground control station, which uses the MAVLink protocol. It provides a simple way of sending commands and receiving status data from the quadcopter. This project mainly uses MAVProxy to override the four RC channels and choose flight mode, as well as gather altitude data.

The custom MAVProxy module implemented during this project, named quadcontrols, provides an additional layer of abstraction to help control the quadcopter. The commands implemented

⁷ For more information about MAVProxy see the official website (Dade, S. n.d.).

and used in this module further abstracts: moving along the horizontal plane; rotating the quadcopter on this plane; and maintaining altitude over time.

The altitude, provided as input from the sonar and gyroscope on the quadcopter combined, is used in conjunction with the calculated position from the image-processing software in order to calculate how the quadcopter should move to get closer to the desired position.

To achieve the regulation, a module named Quadcopter Control, has been developed that implements a proportional-integral-derivative controller (PID-controller) to regulate the relative position to the user. The regulating is done separately for movement along the longitudinal, lateral and normal axes depending on its position in the different axes. Basically the PID-controller calculates how fast the quadcopter should move along these axes depending on its current and previous positions as well as its current speed. To properly find good constants for a PID-controller, a good model for the physics behind how a quadcopter is controlled in form of aerodynamics and external impact is required. As this is complex and not a part of this project, the values for the PID-controller are optimised in a more practical way; by trial and error.

The Quadcopter Control module depends on position input from the Image Processing module and is the last step in the system as it sends maneuvering commands to the quadcopter's autopilot.

5 Imaging

This report speaks of imaging for two subjects. Firstly, the Oculus Rift has been researched and evaluated, this chapter will describe the hardware and highlight some of the important steps. Secondly, the image processing framework OpenCV is applied to the problem of extracting a 3D position for a set of tracking objects within an image.

5.1 Oculus Distortion

The Oculus Rift provides an approximately 100° FOV (Oculus VR, 2012). In order to do so, it magnifies the images with the help of lenses. A by-product of this is that the image suffers from pincushion distortion, causing it to be pinched at the center. This image defect can be corrected by applying barrel distortion, causing a spherised effect and thus neutralizing the defect (see Figure 7). The main purpose of the Oculus Distortion module is to apply said barrel distortion to an image and transfer it to the Oculus Rift.

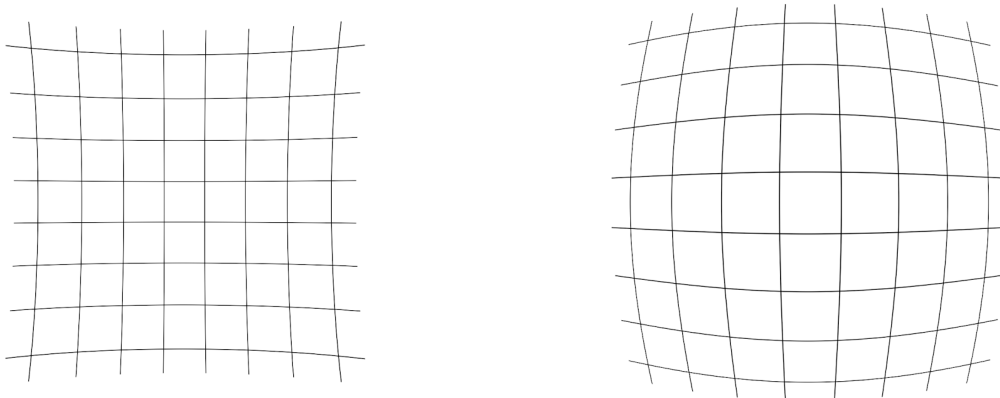


Figure 7. Pincushion distortion (left) and barrel distortion (right)

In order to render the image correctly for the Oculus Rift, the calculated FOV needs to be corrected. Since the lense increases the perceived screen height from $2x$ to $2x'$, see Figure 8 and equation (1), the FOV is calculated from the following:

$$\theta'_{fov} = 2 * \tan\left(\frac{x'}{d}\right) \quad (1)$$

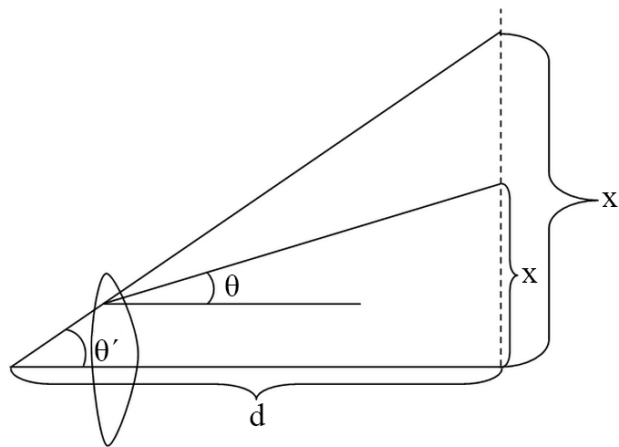


Figure 8. Explaining how a lense changes the FOV

5.2 Image Processing

The goal with the image processing software implemented for this project is to determine a relative position of the camera in regards to the target. As mentioned in earlier chapters, this position is used as the main source of input to the regulation of the quadcopter control.

The calculated position includes depth, which is the distance from the camera to the target, relative height, and lateral distance from the camera to the target. Figure 9 shows the camera's position relative to the target's position P in offsets along the lateral, longitudinal and normal axis. These offsets are denoted as X, Y, and Z, respectively.

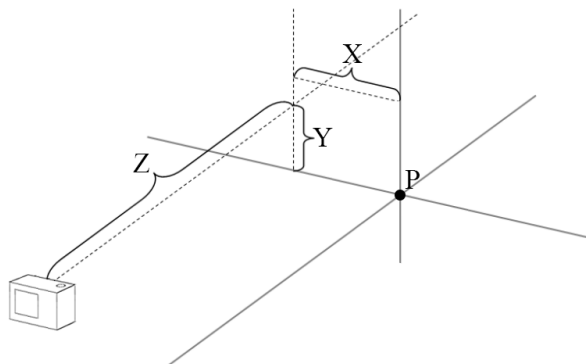


Figure 9. The position of the camera, relative to the target's position P

w

These distances are used to create a 3D vector which will determine the relative position of the target to the camera. By knowing this position of the target, the quadcopter can be regulated to

keep a certain distance to it. To achieve a 3D position of the target, three colourful balls in a triangular pattern are used (see Figure 10). They serve as the *tracking objects*.

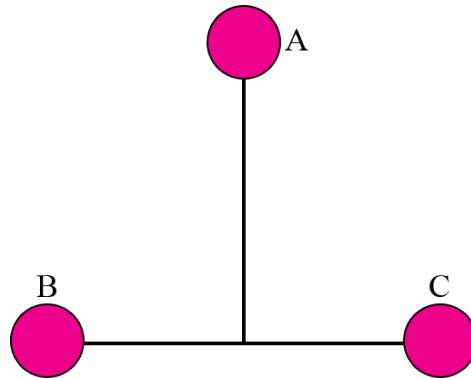


Figure 10. The tracking objects to be tracked

The tracking objects are tracked by colour filtering and circle detection. By having colourful tracking objects, the tracking is made easier as they can easily be distinguished from other parts of the surroundings. Balls are especially useful since they always appear as circles on screen independent of the angle they are viewed from.

The tracking objects are filtered from the camera image and their size and position on screen can be obtained. By knowing the actual size of the tracking objects, their individual distance from the camera can be calculated. This is done using equation (4), which is a combination of equation (2) and (3). Equation (2) describes the relation of the longitudinal distance (denoted as Z) and the width of everything seen on screen, (denoted as W). Equation (3) describes the relation between the width of everything on screen and the width of the screen in pixels (denoted as W_p), utilizing the known relation of the ball on and off screen (b/b_p).

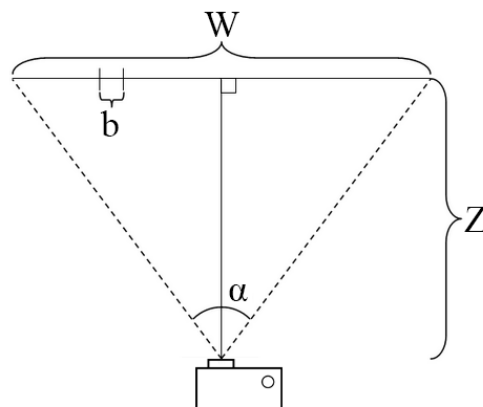


Figure 11. Visualising the trigonometrical relations used to calculate distance

$$Z = \frac{W}{2 * \tan\left(\frac{\alpha}{2}\right)} \quad (2) \quad W = \frac{b * W_p}{b_p} \quad (3) \quad Z = \frac{b * W_p}{2 * b_p * \tan\left(\frac{\alpha}{2}\right)} \quad (4)$$

For full variable explanation, please refer to Table 2 below.

Similarly, the relative height and lateral distance can be obtained through the relation between the screen size of the tracking objects and their actual size, and the distance from the center of the image, as seen in equation (5) and (6).

$$X = \frac{b * \left(b_x - \frac{W_p}{2}\right)}{b_p} \quad (5) \quad Y = \frac{b * \left(b_y - \frac{H_p}{2}\right)}{b_p} \quad (6)$$

For full variable explanation, please refer to Table 2 below.

Table 2. Explanations of position formula abbreviations

| | |
|----------|--|
| W | Width of everything visible to the camera at the distance of the balls |
| W_p | Screen width in pixels, the horizontal camera resolution |
| H_p | Screen height in pixels, the vertical camera resolution |
| b | Actual radius of the balls in centimeters |
| b_p | Radius of the balls in pixels |
| b_y | The y-part of the balls position on screen |
| b_x | The x-part of the balls position on screen |
| α | Angular view of the camera (FOV) |
| X | Lateral position of a ball in centimeters |
| Y | Height of a ball in centimeters |
| Z | Distance to ball in centimeters |

The three tracking objects will, together, form a plane. With a 3D position for all tracking objects, the normal of this plane can be calculated (see Figure 12). Using this normal, it is possible to calculate the relative angle of the target as seen from the camera. This is useful in the case of a third person view, for instance, when the user turns around. When this happens, the quadcopter can change its direction and the camera can capture what the user is seeing.

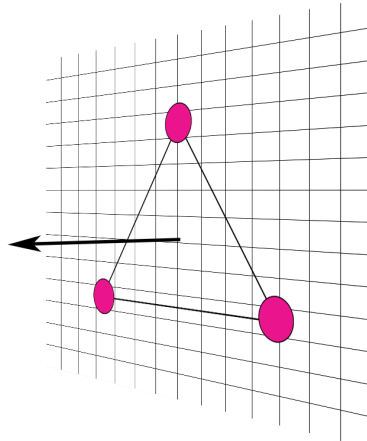


Figure 12. The tracking objects spans a plane and a normal is calculated

5.2.1 OpenCV

OpenCV is used as the base for the Image Processing module. It is a framework that provides algorithms and methods for image processing and computer vision. As these algorithms and methods were already implemented, more time could be spent improving the image processing and tracking of the tracking objects. The main applications of OpenCV in this project is for colour filtering and circle detection.

5.2.2 Target tracking

The tracking developed in this project is done in several steps and uses two colour filtrations and a circle detection. The reason the program uses two colour filtrations is to better distinguish the tracking objects from any possible noise that can occur if the filtering is not perfect, or if other objects in the surroundings have the same colour as the tracking objects. It also makes the software able to find the tracking objects, even if one part of the tracking falters.

Before the colour filtering can be performed, the captured image (see Figure 13) is converted from the RGB colour space to HSV and YCrCb colour space. These colour spaces have independant luma channels (V and Y respectively), which makes the filtering less dependant on

the lighting of the tracking objects (as long as no filtering is done for any of the luma channels). Two different colour spaces is used since the combination is likely to have variation in their noise, which makes noise elimination more effective.

For each colour space, colours that do not match the specified range are filtered out. When visualising this, pixels that match the filter range are coloured white and the ones that are not within the filter range are coloured black, see Figure 14 and 15 for example. Please note that in these figures a noiseless filtering was obtained. If other objects in the surroundings have similar colour as the tracking objects, they will of course be found as objects by the program as well. For each object that is identified by the filter, the software uses OpenCV to find the object's contour which finally can be used to find a center position within the image, as well as the radius of the object.

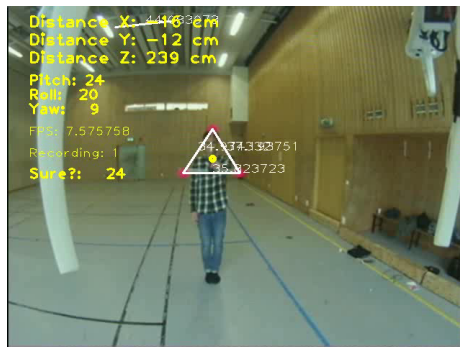


Figure 13. Colour image from camera when the tracking objects has been identified. Please note that the figure includes a lot of debug information.

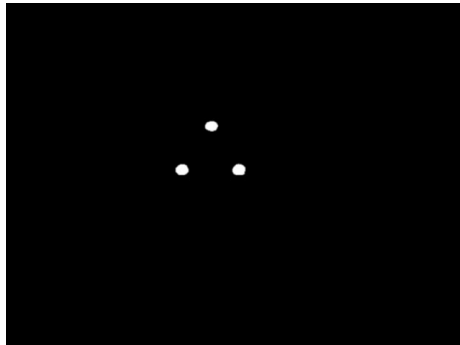


Figure 14. Filtered image using YCrCb colour space

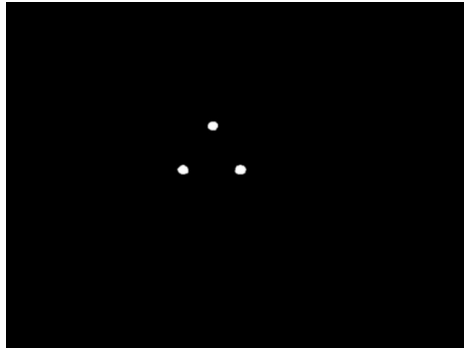


Figure 15. Filtered image using HSV colour space

To further improve the tracking of the tracking objects, circle detection is used. This works by analysing a grayscale version of the captured image as seen in Figure 16 and applying the Hough Circle Detection algorithm that is provided by OpenCV. This works as extra security and will not by itself determine if a circular object is one of the tracking objects or not. It merely increases the priority for the already found objects if the found circle is within a distance and has somewhat the same radius as the colour-filtered tracked objects. In this figure the tracking object is unfortunately not distinguished from the surroundings and do not help the software to become even more sure that the found objects is actually the desired ones.



Figure 16. Grayscale version of image used for detecting circle objects

5.2.3 Priority system

One of the main responsibilities of the the Image Processing module is the priority system. This part of the module is used to determine if any of the identified objects from the camera frame, is one of the targeted tracking objects. The prioritisation algorithm consists of five main steps.

First, the objects found from the two different colour filterings are checked against each other to see if any objects match in both position and size (using the object's radius). The software returns a match if their position is off with no more than half a radius of one of the objects and if the ratio between their radiuses is no more than 1.5. If this matches, the two objects are merged and treated as one object, which gets an increased priority. This new object will be copied into a new list, from here on called L, and used later in the image processing.

When all objects from the two colour filterings have been checked and the matched objects have been merged into one, the objects that did not match against anything are simply transferred into the new list L as well, without any increased priority.

Secondly, the L list is then used in the same matching method against any detected circles coming from the circle detection. This means that if an object also has a circle in the same position and with the same size, that object will get a further increase in priority.

In every loop in the algorithm, the objects with the highest priority in the list L, are saved to be used in in the third step in the priority system. This is done for the last five objects, resulting in a new list containing $5 * 3 = 15$ objects. This list is then matched against L, and the priority for the objects within L is increased if they have matching position and size to any of the objects in the new list. Here, the position and size is less accurately compared than in the previous matching method, since the tracked objects might have moved since the previous frame. The position and size are matched dynamically depending on how old the objects are. The youngest objects have stricter position and radius criteria, which incrementally decreases for older objects. Equation (7) and (8) describes the criterium for position and radius respectively. Note that for equation (7) the same criterium applies analogously for the Y axis.

$$L[i].XPos - L[i].Radius * \frac{5 - index}{2} \leq last[j].XPos \leq L[i].XPos + L[i].Radius * \frac{5 - index}{2} \quad (7)$$

$$\frac{L[i].Radius}{last[j].Radius} \leq 2.5 - 0.25 * index \quad (8)$$

The index i in equation (7) and (8) will go through all objects in L, j will go through the three objects from the $index$ last frame.

The priority system uses the identified objects received from the three filtering and detection steps, and tries to figure out which objects on the screen really are the three tracking objects. This needs to be done since there can be noise left after the various filterings. For example, the circle detection can find many different circles in the image and there might be other objects on the screen with the same colour as the tracking objects.

The fourth step will not actually increase the priority on any objects, but is used in the last, fifth, step. The objects are matched into pairs with the criterium that their approximated distance in 3D space, must match the tracking objects' real distance from each other. The real distance between the tracking objects has been measured to 45 cm between AB and AC and 50 cm between BC (see Figure 10). The module pairs two objects if their approximated 3D distance is between 26 and 52 cm.

Each pair found in the previous steps can be seen as an edge, and the objects as nodes. The fifth and last step in the priority system tries to connect these edges into triangles, which if found further prioritises the nodes in each triangle. The algorithm uses a root node and using its edges, tries to go to all neighbours, and from that, to all neighbours of the visited node, and then from that point back to the root node. If the algorithm succeeds in returning, it can positively identify a triangle (see Figure 17). This will increase the priority on all objects included in the triangle both with a fixed number and dynamically based on the highest priority found in the triangle since before. This can be seen in equation (9).

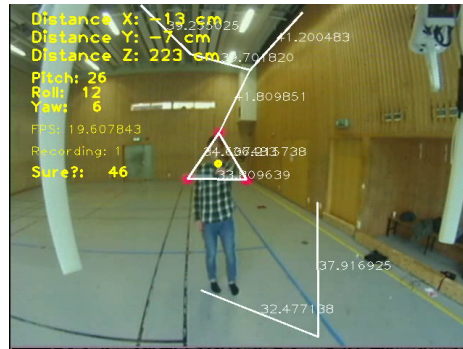


Figure 17. Showing white lines (edges) between objects (nodes) which has been matched into a pair.

$$L[\text{resident}].\text{incPrio} \left(\frac{\text{maxPrio} - L[\text{resident}].\text{Prio}}{2} \right) \quad (9)$$

When the priority system is finished, the three objects with the highest priority are used to determine the relative position of the camera.

5.2.4 Debugging

To help with debugging and analysing the image data and result from the image processing different flags can be set on startup to enable different debug features. For a full list, please see Appendix A III. One very important debug feature is the ability to save all different frames used during runtime as video files using the video flag. This include the original camera feed (in RGB colour, see Figure 13) together with various debug text, depending on the debug flag, both filtered frames (see Figure 14 and 15) as well as the grey scaled frame (see Figure 16). By saving these frames as video files, it is possible to step, frame by frame, to get a better understanding of how the priority system is working and how to better optimise the program to be more sure that the found objects actually is the tracking objects.

6 Testing

To be able to evaluate the performance of the hardware and software, tests were performed. This chapter will highlight how the tests were done and what results were obtained. However, the conclusions are discussed more thoroughly in Chapter 7 and Chapter 8, where appropriate for the context of the results and discussion respectively.

6.1 Quadcopter speed

A speed test was done on the quadcopter to find its top speed and acceleration. By knowing the max speed and acceleration of the quadcopter, the parameters for the regulator could be optimised. These values serve as a good reference to know if the chosen values for the regulator is within a good range, and to get a better understanding of what the chosen regulator values will result in for the quadcopters speed. The maximum speed was roughly measured to 15.4 m/s. For the full speed test, please see Appendix B.

6.2 Video latency

To improve the user experience and the quadcopter control, a low latency on the video is desired. Tests were made to evaluate whether or not the latency could be problematic. The test technique used was very simple. Each test included the GoPro camera filming a stopwatch and a display showing the output, either a TV or a PC using basic video playback software. A second camera was then used to take a photo of the stopwatch and the display. The delay of the third-person view setup was interpreted as the difference in time of the stopwatch and the displays view of the stopwatch, see equation (10).

$$R = T_{stopwatch} - T_{display} \quad (10)$$

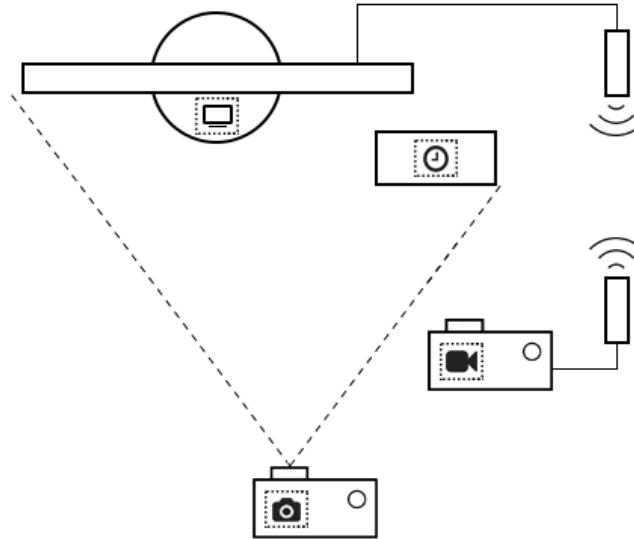


Figure 18. Test setup, a camera is capturing the display and PC screen at the same time

To achieve accurate results, each test was performed several times and an average of the measurements was calculated.

This procedure was repeated with different equipment between display and the GoPro, to analyse what impact different hardware had on our solution. Noteworthy is that while Figure 18 only shows a wireless connection between the GoPro and the display, solutions with a cable connection was tested this way as well.

6.2.1 Test data

Some latency was experienced during the transmission of the video stream to the computer and application of the barrel distortion to the frames. The following latencies were measured on the final product:

Table 3. Summary of video latency test data

| Test | Devices and software | Delay | Deviation |
|-------------|---|--------------|------------------|
| 1 | Camera to grabber, using Cam-Share | 190.4 ms | 11.36 ms |
| 2 | Camera to grabber, using Cam-Share and Oculus Distortion | 235.2 ms | 14.96 ms |
| 3 | Camera to grabber wirelessly, using Cam-Share | 208.3 ms | 15.38 ms |
| 4 | Camera to grabber wirelessly, using Cam-Share and Oculus Distortion | 204.9 ms | 15.92 ms |
| 5 | Camera to TV | 167.4 ms | 18.08 ms |
| 6 | Camera to TV wirelessly | 164.7 ms | 14.44 ms |

The most interesting data is the delay when sending the video wirelessly to the grabber while using Cam-Share and Oculus Distortion, as this is what will be used in the full system. The rest of the data shows that the biggest delay seems to come from the camera itself. According to a white paper on latency (Cast, 2013), an acceptable delay in video, when playing for example video games is below 100 ms as this will not be noticeable in most cases. Since the worst case from the measured data is 235 ms the system does not have acceptable latency. Of Course acceptable latency is subjective and dependant on what the video is used for.

6.3 Vibration

To ensure that the quadcopter performed as good as possible and did not make any unnecessary disturbances that could affect other equipment, a vibration test was performed. This showed (see Figure 19) that the quadcopter was well within the optimal limits for the accelerometer vibrations.

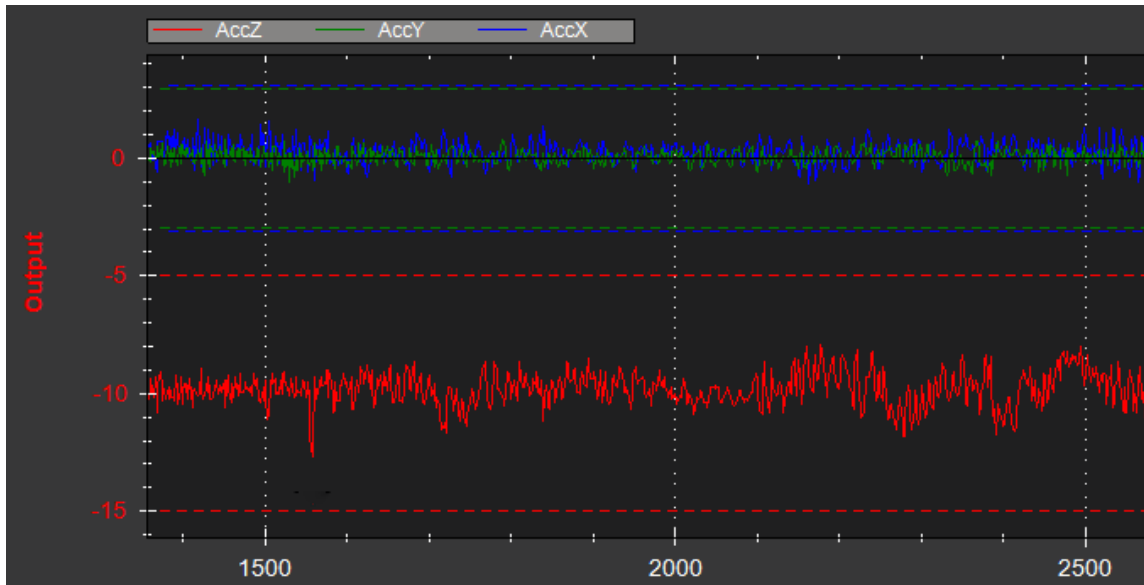


Figure 19. Graph of vibrations experienced by accelerometer

Figure 19 shows the alterations in acceleration during time. the accelerations are measured in the longitudinal, lateral and normal axes, shown as AccY, AccX and AccZ respectively in the figure. As seen AccZ lies around -10 due to the gravitational pull of the earth. According to the Ardupilot homepage on measuring vibration (Ardupilot, 2014) these values are acceptable when $-3 < \text{AccX} < 3$, $-3 < \text{AccY} < 3$, $-5 < \text{AccZ} < -15$.

7 Results

A system has been designed that creates an immersive experience by letting a user view itself from a third-person perspective. Software to control the quadcopter to follow the user around has been implemented, as well as software to receive a wireless camera feed. The control is regulated through image processing of the camera feed, and the camera feed is barrel distorted to match the display correctly on the Oculus Rift.

7.1 In-flight video transmissions

The system has been developed by a process of hardware design decisions that were discussed in Chapter 2, complemented with a software implementation described in Chapter 3. The project has come a long way in reaching its goals in implementing a third-person view.

The first milestone that presented results relating back to the purpose of this thesis, was the first successful in-flight transmission of video. This milestone was reached through an experiment to evaluate how the transmitted video would appear in the Oculus Rift, in terms of quality. The figure below shows a frame of the video as it was stored in the shared memory segment by the Cam-Share module.



Figure 20. Video frame from first in-flight video transmission experiment

There are two comments that should be made on the quality of the video feed that the frame in Figure 20 is captured from. Firstly, the low resolution; and secondly, the visible horizontal line distortions. These are both consequences of using analog video transmission, but as discussed in Section 2.3, the requirement of a low latency on the video feed favors analog over digital video. Therefore, analog video was kept as the favored format for video transmission.

7.1.1 Immersive user experience

This experiment made it possible to relate back to the purpose of assessing the immersive quality of the system. Setting quadcopter positioning aside, this experiment was an indication on the effects on user experience from both the predicted low image resolution as well as video latency.

The effects of the low image resolution were not as dampening to the experience as one could believe. During high speed maneuvering of the quadcopter for example, the experience was rather immersive in the sense that it was easy to forget about your actual surroundings.

However, combined with the latency it was quite clear that what you saw inside the Oculus Rift glasses was merely a recorded video of yourself moving around, rather than an out of body experience.

7.2 Attempting autonomous control through image processing

As all of the core package modules described in Chapter 3 about software implementation relied on a video feed, the transmission experiment from Section 7.1 was a major milestone for the project. It meant a fair chance of also being able to process this image using the Image Processing module. Figure 21 displays one of the experiments where the Image Processing module is activated and tries to locate the target inside the video. The same figure also introduces the largest issue found so far with analog video streams from the quadcopter.

Apart from using the streamed video as input to the Oculus Distortion module, the project team had to evaluate how it could be used as input to the Image Processing module. In order to evaluate this, an experiment with two separate phases was conducted.

Phase one of the experiment simply lets the Image Processing module analyse the transmitted video stream and output the relative position of the user (along with other debug information). As seen in the left image of Figure 21 below, a feasible distance in all three axes is printed on screen. This phase was considered a success.

Phase two was carried out similarly to phase one, but with the modification that the position was

used as input to the Quadcopter Control module. What this means in practice, is that the system running uses the position as input to regulate the radio control of the quadcopter. As seen by the broken white triangle in the right image of Figure 21 below, this phase was not successful. Noise in the image prevents the Image Processing module from acquiring a stable position of the user, which in turn has the effect on Quadcopter Control that it refuses to take control of the quadcopter.

As Quadcopter Control utilises the 3DR radio set to send control commands to the quadcopter, it is a fair guess that this hardware might be what is causing disturbances in the image. A couple of attempts to reduce the disturbance was made by moving the 3DR radio set and the video transmitter to opposite sides of each other, so that the signals and electromagnetic fields generated by the current should affect each other as little as possible. However, these attempts were rather futile as no significant decrease in disturbance could be seen.

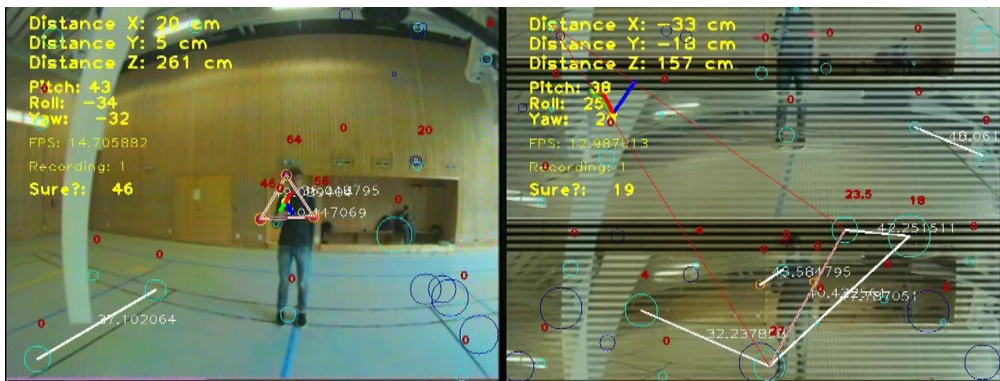


Figure 21. Image without disturbances (left), Image with disturbances (right)
The circles and lines drawn represent the objects that are being identified in the algorithm

7.3 Image processing evaluation

Part of the purpose of this thesis was to evaluate whether or not image processing is a good way of implementing autonomous control of airborne vehicles. From the previous results and tests, it can be concluded that using image processing to implement autonomous control has a few problems, mainly image disturbances and latency during video transmission. Although, when using the camera without radio transmission, the Image Processing module could determine the position of the user in a way that should be sufficient for autonomous flight.

8 Discussion

It was apparent from the beginning that assembling a quadcopter which autonomously follows a user would turn out to be a challenging task, requiring a lot of research and planning. The final solution required a combination of several different technologies, such as image analysis, control systems engineering and virtual reality.

Chapter 7 presents two major blocks that the results can be categorised within, namely image processing and video presentation to the user. The first category has been tougher to succeed with and the latter has been easier. Sections 8.1 and 8.2 will discuss the image processing category whereas video presentation will be discussed in Section 8.3. The last sections will discuss other important aspects that have either helped or delayed the progress as well as how the system can be improved on a more visionary scale.

8.1 Negative aspects of using image processing for autonomous control of UAVs

The results in Chapter 7 presents the positive and negative aspects of controlling a quadcopter autonomously using image processing. Two major downsides can be distinguished. The first one by looking at the test results from Section 6.2, video latency. The second one by looking at Section 7.2, image noise.

8.1.1 Video latency analysis

The latency from the video transmission was a lot higher than desired and expected. As seen in Table 3 from Section 6.2.1, the biggest culprit for the latency was the GoPro camera. Aside from the camera, the largest delay came from the frame grabber used to convert the analog signal to a digital signal. For latency optimisation, the main section to improve would be the camera and frame grabber device. The average latency using only the GoPro camera connected via an analog cable to a display was 167.4 ms, with a deviation of 18.08 ms. The other components combined only increased it by an additional average of 67.8 ms.

As no expertise in analog video and the transmission of it was obtained prior to the project, much research is still left on the subject, thus a possible solution to the latency issue could be to investigate further which hardware should be used for image capturing, as well as transmission. This in turn could lead to the user of the system seeing her own movements earlier, thus increasing the immersive experience.

8.1.2 Image noise analysis

As noise in analog signals are common, some noise was tolerated in the system during development, as long as it did not disturb the Image Processing. This type of noise was also present during normal flight using an RC controller.

However, the video disturbance during autonomous flight posed a huge problem in the Image Processing module. One part of the distortion was that the image went black and white from time to time. This was a problem since one of the main properties the image software utilised when analysing the image was the colour of the tracking objects. This also made the backup feature, to detect circles, useless since detected circles could not be matched to any filtered objects. The distortions also made the image move around and added black bars randomly or switched the top and bottom part of the image completely which prevented the image-processing software to work as intended.

An option to reduce the problem affecting the regulation caused by the disturbances would be to do the calculation on the quadcopter, rather than on a ground control station. However, it is questionable whether or not a computer with low enough power consumption and yet high enough processing power to do this can be easily obtained. Putting further strain on the battery used to power the APM and the motors would reduce the air-time per battery as well. This solution would however be quite reasonable for larger UAVs, drones used in military operations for example.

Let us assume that the observation presented in Section 7.2 is true, namely that the cause of the image noise making autonomous flight fail is the signals from the radiolink. This makes one apparent solution, to try to isolate the video transmission from the source of the disturbing signals. For example, manually by trying setups with different placements of all pluggable components, such as the radiolink and the video transmitter itself. However, this was evaluated during the test flights and no noticeable improvements were observed. Further experiments, using for example aluminium to shield the video link from disturbances is a natural next step that could be taken. This was considered a rather complex experiment due to how it could affect the aerodynamics of the quadcopter, and therefore not prioritised within the time limit of the project.

It might seem simple to have switched to another camera setup and transmit the video digitally. However, the gimbal used during this project was manufactured and optimised for use with a GoPro camera, which in turn has a high latency for digital video over its WiFi. This limits the choice of cameras that can easily be mounted on the quadcopter while still being able to stabilise the camera as the quadcopter moves around.

8.2 Positive aspects of using image processing for autonomous control of UAVs

Image processing can be a viable method for autonomous control, if the two downsides discussed in the two previous sections can be negated. The motivation is based on the fact that this project, which was carried out without prior experience within the area of image processing, managed to accomplish feasible results from the Image Processing module with limited resources. The positive results from using image processing rather than other methods is that no additional non-application critical components besides the camera is needed on the quadcopter. This reduces the total weight and complexity of the quadcopter. These results were presented in Section 7.2.

8.2.1 Predicting movements using previous data

An example of a large scale project that heavily relies on image processing as input to autonomous control of vehicles is Google Cars. They have successfully been able to navigate cars in real city traffic by using image processing (Madrigal, 2014). According to Madrigal there are two major reasons for why the Google Cars project manages this. Firstly, they use additional sensors to cameras (as well as a full rig of cameras instead of just one) to process the world around them. Secondly, in addition to process their surroundings in real time, they also utilise an extremely detailed 3D model of the surroundings which has been generated from vast amount of previously gathered data.

Relating back to this thesis. Using data from previous flights as Google Cars is doing with their drives, and recording flight experience from the system, could prove to be an interesting way of preventing collisions with other objects than the one(s) tracked.

8.3 Implementing a third-person view in real life

Two key factors for getting a pleasant display of oneself is easily argued as clear and current, which can be translated into high resolution and low latency.

The components that are responsible for capturing, transporting, and presenting the video in the system implemented for this thesis, is the Oculus Distortion module, the Oculus Rift, the GoPro, and the video transmission hardware. As presented in Section 2.3 the video resolution presented to the user on the displays of the Oculus Rift is quite low and has quite some negative critique. Furthermore, the use of analog video transmission prevents the system as it is implemented today from using a higher actual resolution than this.

Combine the low resolution cap with how digital video, which is discussed as a candidate for video transmission in Section 2.3, but then discarded due to the expected high latency points to the conclusion that alternative approaches must be made. Or maybe even that further development on hardware is still very much in need of development before an ideal third-person view in real life can be implemented.

8.4 Quadcopter speed and vibration tests

Two quadcopter tests were performed to evaluate the performance of the UAV, a speed test and a vibration test.

The speed test from Section 6.1 was used as a reference for the regulation of the quadcopter. The test also shows that the product is not limited by the speed of the quadcopter, a person could run as fast as desired and the quadcopter would be able to keep up in a straight line, although turning to change direction could still be a limiting factor. The high maximum speed of the quadcopter also allows the quadcopter to follow for example cars and other vehicles, as well as skiers and bicyclists.

As described Section 6.3 the vibration test results are well within the optimal range. From this, it can be concluded that the vibration dampening was well implemented and the vibration did not cause any problems affecting the flight of the quadcopter.

8.5 Autopilot board damage

Due to lack of understanding from the project group and lack of documentation from the open source community, there was a major setback during the project when attempting the final assembly steps, i.e. when connecting the APM board to a laptop for firmware installation and configuration. The problem faced was that the APM's external sensor circuit was extremely sensitive to low voltage⁸, which was exactly what it was exposed to when powering the board with a USB source. This resulted in a damaged APM unable to connect to any external sensors and further configurations were impossible.

Once proper research had been done on this issue, a functional APM was used and the configuration and real flight testing could begin.

Due to the APM board problems, all the software had to be tested on a software-in-the-loop simulation of a quadcopter. This caused a major delay in the project and prevented proper testing of the software on a real quadcopter. This caused further problems like the picture disturbance to be discovered very late in the project, resulting in this issue still being unsolved.

⁸ For more information about the APM board low voltage problem see (Mackay, 2013)

8.6 Future development potential

To improve the user experience further and provide a more immersive feeling, the head tracking data from the Oculus Rift should be gathered and used together with the image-processing and regulator software when moving the quadcopter. The gyroscope of the Oculus Rift could determine if the user moved his/her head to the left or to the right and let the quadcopter move correspondingly. By letting the quadcopter go to the right or left, and at the same time yaw counter clockwise or clockwise, it will result in an arc motion (see Figure 22), which would mimic how the user actually moved his/her head when looking in a certain direction.

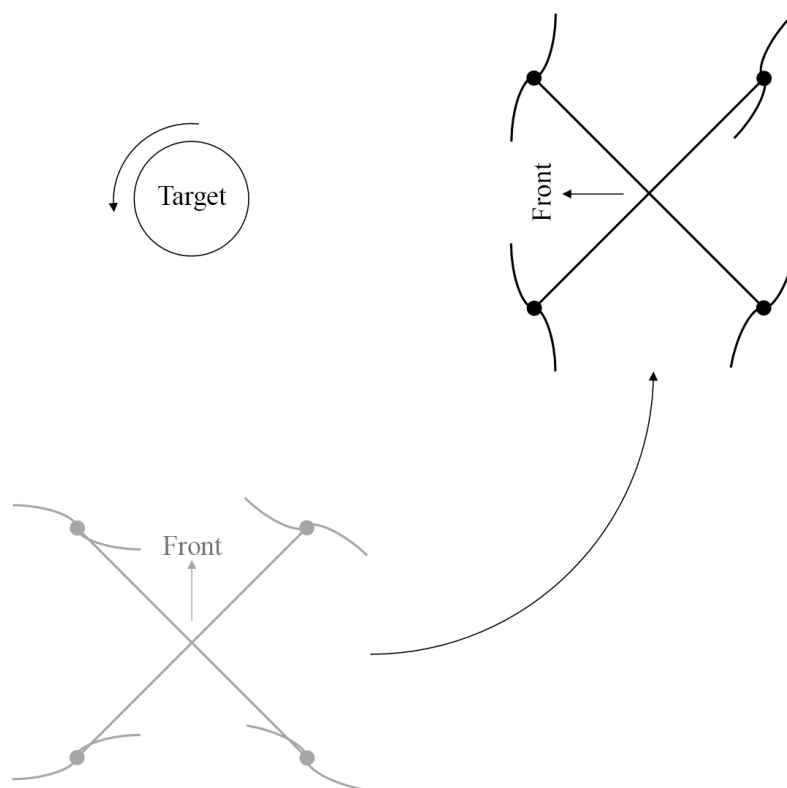


Figure 22. Example of rotational implementation of the quadcopter

Further, the system could create immense virtual realities with immersive environments, which could be used to create games or other entertainment projects where users can see themselves as the main character with the third-person view. Virtual reality experiences with the Oculus Rift has been done before, but rarely in a way where the users see themselves in the real world with a partially virtual overlay.

As stated under chapter 1.3 delimitations, the quadcopter could be equipped with ultrasonic sensors on its sides to be able to detect walls. This would result in a safer flight and a lower risk of impact with the surroundings. If the target rotates while walking beside a wall or other obstacles, the quadcopter could notice that something is approaching it too close (see Figure 23) and abort its movement in the direction to avoid a crash and instead move, for example, only forward to continue follow the target.

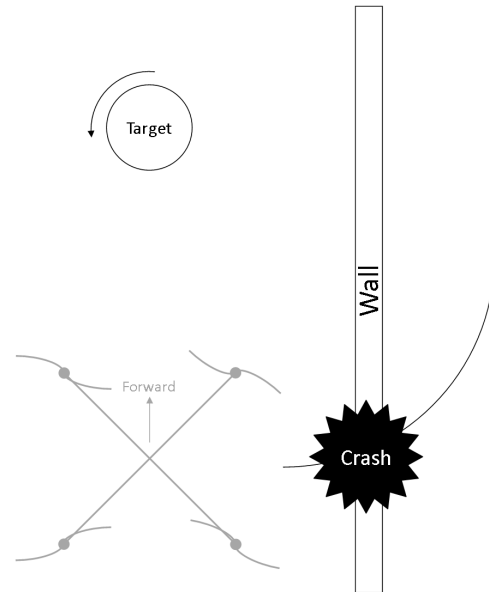


Figure 23. Quadcopter not being aware of its surroundings can cause crashes if the target is not aware of its movements in regard to the surrounding.

8.6.1 Real-world applications

After accomplishing autonomous control of the quadcopter, such that it is able to safely and precisely follow the user, adding additional commands that the user is able to send to the quadcopter, such as regulating heights or scouting the environment, could prove to have interesting applications to real problems people are facing. Missing people could be sought for in narrow or otherwise dangerous places, and policemen could get a bird eye view that does not require any additional personnel in order to maneuver the field of vision.

The hardware used in this project makes it possible to add additional sensors and other accessories such as GPS, which could increase the quadcopter's ability to fly autonomously with waypoints and, for example, be able to map a disaster site by flying to different waypoints and give the user a better understanding of their surroundings.

While this system as a whole may not be easily applied to these kinds of situations, parts of it is. Video transmission with low latency and autonomous control of UAVs based on data originating from the vehicle are both areas that could help development of systems that could really help in these kind of situations.

8.7 Ethics

UAVs with the possibility to record or transmit video can cause several ethical implications. By using normal means, such as a fence or wall, people can easily be restricted from private areas, however, a ground based perimeter can not hinder an unauthorised UAV. With greater accessibility to mobile remote-controlled vehicles, the privacy could be disturbed more easily. In countries where it is legal for people to take pictures of private areas from a public area, this may become a problem. The scenario where a quadcopter takes a sneak peak through the fifth floor window is possible and would be legal given that the pilot has a permit for flying the quadcopter.

9 Conclusions

This thesis proposed a vision-based autonomous target tracking system for a UAV that can be used to present a third-person view of any kind of object. Image processing as a way of autonomously controlling a quadcopter was analysed and proved to be a somewhat viable solution. However, issues in video transmission were found. Due to these issues mainly, a complete system as the one stated in the objective of this thesis could not be developed to its full extent. Other than that, all important milestones were reached, and the project is seen as successful. Hopefully, future works will be devoted to carry on this project to fulfill its objective. Ways of visually detecting a target without the use of external objects, such as balls, can be evaluated, and faster video transmission hardware can be used to improve data transfer between the quadcopter and the ground station.

References

- Ardupilot (2014) Measuring Vibration. *AC_MeasuringVibration* | *ArduCopter*.
http://copter.ardupilot.com/wiki/ac_measuringvibration/ (2014-05-19)
- Azrad, S., Kendoul, F., Nonami, K. (2010) Visual Servoing of Quadrotor Micro-Air Vehicle Using Colour-Based Tracking Algorithm. *Journal of System Design and Dynamics*, vol. 4, nr 2, pp. 255-268.
- BOSCAM TS 351 - RC 805 Manual. (n.d.)
<http://www.hobbyking.com/hobbyking/store/uploads/434487232X422882X24.pdf> (2014-05-19)
- Bresciani, T. (2008) *Modelling, Identification and Control of a Quadrotor Helicopter*. Lund: Lund University (Master Thesis, Department of Automatic Control).
- Cast (2013) White Paper: Understanding—and Reducing—Latency in Video Compression Systems. Cast. <http://www.cast-inc.com/blog/white-paper-understanding-and-reducing-latency-in-video-compression-systems> (2014-06-02)
- Dade, S. (n.d.) MAVProxy. <http://tridge.github.io/MAVProxy/> (2014-05-16)
- Deyle, T.J. (2005) *Low-Cost Inertial Measurement Unit*.
- Hatchet. (n.d.) <http://www.jurmol.com/uav.html> (2014-05-17)
- Höflinger, F. et al. (2013) A Wireless Micro Inertial Measurement Unit (IMU). *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT*, vol. 62, nr 9, ss. 2583-2595. (2014-05-17)
- Gancarz, M. (2003) *Linux and the Unix Philosophy*. Elsevier Inc.
- Orland, K. (2013) Impressions: Oculus Rift dev kits ready to bring virtual reality to the masses. Arsthecnica. <http://arstechnica.com/gaming/2013/04/impressions-oculus-rift-dev-kits-ready-to-bring-virtual-reality-to-the-masses/> (2014-04-18)
- Mackay, R. (2013) AC3.1 & blown 3.3V regulator (was AC3.1-rc5 SPI speed problem)
<http://diydrones.com/forum/topics/ac3-1-rc5-spi-speed-problem> (2014-05-16)

Madrigal, A. (2014) The Trick That Makes Google's Self-Driving Cars Work. The Atlantic. <http://www.theatlantic.com/technology/archive/2014/05/all-the-world-a-track-the-trick-that-makes-googles-self-driving-cars-work/370871/> (2014-05-19)

martcerv. (2013) Wifi video Lag test GoPro App v2 hero3 black HD3.03.03.00 Galaxy s4. [YouTube]. <https://www.youtube.com/watch?v=89IIIjWJhE> (2014-05-13)

MAVLink Micro Air Vehicle Communication Protocol - QGroundControl GCS. (n.d.) <http://qgroundcontrol.org/mavlink/start> (2014-05-16)

Niroumand, F-J., Fakharian A. and Seyedsajadi M-S. (2013) Fuzzy Integral Backstepping Control Approach in Attitude Stabilization of a Quadrotor UAV. In *Iranian Conference on Fuzzy Systems (IFSC)*. Aug, 2013.

Oculus VR (2012) Update on Developer Kit Technology, Shipping Details. <http://www.oculusvr.com/blog/update-on-developer-kit-technology-shipping-details/> (2014-11-28)

Oculus VR (2014) Oculus Rift, Next-Gen Virtual Reality. *Oculus Rift - Virtual Reality Headset for Immersive 3D Gaming*. <http://www.oculusvr.com/rift/> (2014-05-13)

Oculus VR (2014) DK2, Development Kit 2. Oculus VR. *The All New Oculus Rift Development Kit 2 (DK2) Virtual Reality Headset*. <http://www.oculusvr.com/dk2/> (2014-05-13)

OpenCV (2014) About. *ABOUT | OpenCV*. <http://opencv.org/about.html> (2014-05-14)

pipe(7) - Linux manual page. (2005) <http://man7.org/linux/man-pages/man7/pipe.7.html> (2014-05-16)

remote-eyes - Videograbber to Oculus Rift for FPV - Google Project Hosting. (n.d.) <https://code.google.com/p/remote-eyes/> (2014-05-16)

shm_overview(7) - Linux manual page. (2010) http://man7.org/linux/man-pages/man7/shm_overview.7.html (2014-05-16)

Solon, O. (2013) Iranian robotics lab works on ocean rescue drone. *Wired*, 27 Mars 2013.

Scrum Alliance (n.d.) Why Scrum? *What is Scrum? An Agile Framework for Completing Complex Projects - Scrum Alliance*. <http://www.scrumalliance.org/why-scrum> (2014-05-16)

| | |
|--|-----------|
| Appendix A - Software Documentation | 2 |
| <i>I. Cam-Share</i> | 2 |
| Description | 2 |
| Init | 2 |
| Read | 2 |
| <i>II. Third-person-remote-eyes</i> | 4 |
| Description | 4 |
| Usage | 4 |
| <i>III. Image Processing</i> | 5 |
| Description | 5 |
| Usage | 5 |
| <i>IV. Quadcopter control</i> | 8 |
| Description | 8 |
| Usage | 8 |
| Appendix B - Test Results | 10 |
| <i>I. Latency Test Data</i> | 10 |
| Test Data | 10 |
| Summary | 12 |
| <i>II. Quadcopter Speed Test</i> | 13 |
| Appendix C - Hardware | 14 |
| <i>I. Quadcopter</i> | 14 |
| AutoPilot - ArduPilot Mega (APM) 2.6 | 15 |
| <i>II. Oculus Rift</i> | 16 |
| <i>III. Remote Controller</i> | 16 |
| <i>IV. Radio Receiver</i> | 17 |
| <i>V. Charger</i> | 17 |

Appendix A - Software Documentation

This document will describe the interfaces between different modules (separate applications) within the core of the Third Person Immersion software.

I. Cam-Share

Github: <https://github.com/third-person-immersion/cam-share>

Input:

- Reads the video input stream from camera

Output:

- Writes a textfile "caminfo.log" on the format "***width\nheight\nmemorySize\n***"

Description

The Cam-Share application reproduces the video stream by fetching the frames from the video stream and place them in a shared memory space. This enables multiple processes to simultaneously access the video stream.

The module is split into two parts as follows:

Init

Starts writing to the shared memory segment

Binary: `./init`

Read

Binary: `./read` (NOTE: This binary is not needed for the system, it is merely a test of the Read class)

The class ***Read_<os>.h*** can be included in external applications.

Use **`Read.getFrame().copyTo`** for getting your own modifiable frame.

Note: no reading application may write to the shared memory. Modifications must be made on copied frames.

Command line arguments

USAGE:

```
./init [-l <string>] [-c <int>] [--] [--version] [-h]
```

Where:

-l <string>, --log <string>
Select log file

-c <int>, --camera <int>
Select camera

--, --ignore_rest
Ignores the rest of the labeled arguments following this flag.

--version
Displays version information and exits.

-h, --help
Displays usage information and exits.

Command description

Writes video input to the shared memory segment

II. Third-person-remote-eyes

Github: <https://github.com/third-person-immersion/third-person-remote-eyes>

Input:

- Reads video frames Shared Memory using identifier **2581** (Cam-Share output)

Output:

- Produces a barrel distorted video stream on screen viewable on the Oculus Rift (positioning of the video is up to OS)

Description

Third-person-remote-eyes is a forked project¹ which produces a distorted image for the Oculus Rift and have been modified to fetch video from the shared memory.

Usage

Press F9 to go fullscreen press Pos1/Home or End to change picture size.

¹ remote-eyes: <https://code.google.com/p/remote-eyes/>

III. Image Processing

Github: <https://github.com/third-person-immersion/image-processing>

Input:

- Reads video frames Shared Memory using identifier **2581** (Cam-Share output)

Output:

- Vector describing the 3D position of the user (triangle) from the Quadcopter (camera)
- Vector holding the angles of the user (triangle)
 - Yaw
 - Pitch
 - Roll

Description

Image processing reads video from the shared memory space and performs filtering to triangulate and decide the location of an object in 3D space relative to the camera.

Usage

Output to the console if -r flag is set, meaning release. Also the output will start with a error output to check if everything regarding the camera opening etc worked OK.

rflag = 1:

- Only distance from the camera (Z distance)

rflag = 2:

- All above
- Full 3D position (X, Y and Z distance)

rflag = 3:

- All above
- Full angle vector
- Sureness (How accurate the program is that the objects found really is the tracking objects)

For more details of ASCII, please see <http://www.asciitable.com/>

The 3D position is considered one group, the angles another and sureness value yet another. Inside every group there will be a maximum of three (3) records (e.g. X, Y and Z position in 3D position group). To separate these the ASCII group separator (char GS, Dec 29) will be used. To separate the records the ASCII record separator (char RS, Dec 30) will be used. Using these dedicated separators, we ensure not to separate our data faulty.

Please note! To be able to display the format of the output, <GS> and <RS> will be used where the group/record separator will be used! The output is formatted as:

X-DISTANCE<RS>Y-DISTANCE<RS>Z-DISTANCE<GR>X-ANGLE<RS>Y-ANGLE<RS>Z-ANGLE<GR>SURENESS

Example where rflag is set to 1:

202.123412

The value 202.123412 says that the object is roughly two meters from the camera

Example where rflag is set to 3:

20.123<RS>11.441<RS>187.3123<GR>-35.192<RS>41.333<RS>0.312<GR>0.63

Here there are three values in each group except the last group (and all three groups are present). The last value indicates that the program is 63 % sure that the found objects really is the tracking objects.

Parsing:

Parsing the output will be done by reading, line by line, and splitting the string first using <GR> character. Then by checking the length of the new array, one can determine how many values are present. If length is 1, then only 3D position is present, if length is 3, then 3D position data, angle data, and sureness data is present.

Command line arguments

USAGE:

```
./image-processing [-s <int>] [-r <int>] [-f <string>] [-v <string>]  
                  [-d <int>] [--] [--version] [-h]
```

Where:

-s <int>, --distance <int>
Set the desired distance

-r <int>, --release <int>
Activate release mode

-f <string>, --logfile <string>
Set log file path

-v <string>, --video <string>
Save video

-d <int>, --debug <int>
Activate debug mode

--, --ignore_rest
Ignores the rest of the labeled arguments following this flag.

--version
Displays version information and exits.

-h, --help
Displays usage information and exits.

Image processing reads video from shared memory space and performs filtering to triangulate and decide location of an object in 3D space relative to the camera.

IV. Quadcopter control

Github: <https://github.com/third-person-immersion/quadcopter-control>

Input: A 3D vector from image-processing describing the distance from the user to the quadcopter

Output: A mavlink signal readable by the 3DR radio set, steering the quadcopter closer to the desired location

Description

The Quadcopter controller is used to receive the relative distance to the user from the Image-processing software and with this data calculate how the quadcopter should move and in which direction.

Usage

A ground control station called MAVProxy is a prerequisite for Quadcopter controller to work. A module called quadcontrols is also needs to be loaded into Quadcopter controller by using the --load-module start parameter.

Quadcontrols module

Github: https://github.com/third-person-immersion/MAVProxy/blob/master/MAVProxy/modules/mavproxy_quadcontrols.py

A module for the MAVProxy ground control station used by Quadcopter controller. The module adds commands to move a quadcopter in various ways.

Command line arguments

USAGE:

```
./regulator [--master] [--baudrate <int>] [--load-module] [--quadcopter]
```

Where:

--master

Specifies which port (serial, USB or network address) the UAV uses for communication

--baudrate <int>

Sets the baudrate for 3DR Radio

--load-module

Load a specific module on startup

--quadcopter

Enables quadcopter controls

Quadcopter Control regulates a Quadcopter using MAVProxy with help of a 3D vector containing the current relative location to the user.

Appendix B - Test Results

I. Latency Test Data

Test data measuring the latency of setup. In total, 6 tests were performed to measure the delay of different parts of the system to find the cause of delay.

Test Data

Test1 (Camera to grabber, using cam-share and read)

| # | Display | Camera | Delay | Deviation |
|-----------------|---------|--------|--------------|--------------|
| 1 | 485344 | 485163 | 181 | 9,4 |
| 2 | 482915 | 482702 | 213 | 22,6 |
| 3 | 470711 | 470529 | 182 | 8,4 |
| 4 | 466370 | 466188 | 182 | 8,4 |
| 5 | 462361 | 462178 | 183 | 7,4 |
| 6 | 460567 | 460385 | 182 | 8,4 |
| 7 | 456827 | 456645 | 182 | 8,4 |
| 8 | 455249 | 455065 | 184 | 6,4 |
| 9 | 454096 | 453883 | 213 | 22,6 |
| 10 | 407734 | 407532 | 202 | 11,6 |
| Average: | | | 190,4 | 11,36 |

Test2 (Camera to grabber, using cam-share and Oculus Distortion)

| # | Display | Camera | Delay | Deviation |
|-----------------|---------|--------|--------------|--------------|
| 1 | 6732 | 6519 | 213 | 22,2 |
| 2 | 7173 | 6900 | 273 | 37,8 |
| 3 | 6922 | 6684 | 238 | 2,8 |
| 4 | 5244 | 4999 | 245 | 9,8 |
| 5 | 8171 | 7933 | 238 | 2,8 |
| 6 | 7100 | 6893 | 207 | 28,2 |
| 7 | 5727 | 5484 | 243 | 7,8 |
| 8 | 5906 | 5677 | 229 | 6,2 |
| 9 | 1450 | 1201 | 249 | 13,8 |
| 10 | 4344 | 4127 | 217 | 18,2 |
| Average: | | | 235,2 | 14,96 |

Test3 (Camera to grabber wirelessly, using cam-share and read)

| # | Display | Camera | Delay | Deviation |
|-----------------|---------|--------|--------------|--------------|
| 1 | 52502 | 52290 | 212 | 3,7 |
| 2 | 49462 | 49280 | 182 | 26,3 |
| 3 | 43848 | 43636 | 212 | 3,7 |
| 4 | 41270 | 41057 | 213 | 4,7 |
| 5 | 28164 | 27932 | 232 | 23,7 |
| 6 | 19414 | 19203 | 211 | 2,7 |
| 7 | 13365 | 13182 | 183 | 25,3 |
| 8 | 11326 | 11143 | 183 | 25,3 |
| 9 | 9077 | 8833 | 244 | 35,7 |
| 10 | 4304 | 4093 | 211 | 2,7 |
| Average: | | | 208,3 | 15,38 |

Test4 Camera to grabber wirelessly, using cam-share and Oculus Distortion

| # | Display | Camera | Delay | Deviation |
|-----------------|---------|--------|--------------|--------------|
| 1 | 9046 | 8835 | 211 | 6,1 |
| 2 | 5234 | 5050 | 184 | 20,9 |
| 3 | 2728 | 2541 | 187 | 17,9 |
| 4 | 5938 | 5724 | 214 | 9,1 |
| 5 | 3870 | 3622 | 248 | 43,1 |
| 6 | 1398 | 1186 | 212 | 7,1 |
| 7 | 8567 | 8382 | 185 | 19,9 |
| 8 | 4298 | 4083 | 215 | 10,1 |
| 9 | 1554 | 1370 | 184 | 20,9 |
| 10 | 4782 | 4573 | 209 | 4,1 |
| Average: | | | 204,9 | 15,92 |

Test5 (Camera to TV)

| # | Display | Camera | Delay | Deviation |
|-----------------|---------|--------|--------------|--------------|
| 1 | 78019 | 77836 | 183 | 15,6 |
| 2 | 78688 | 78536 | 152 | 15,4 |
| 3 | 78840 | 78658 | 182 | 14,6 |
| 4 | 78962 | 78780 | 182 | 14,6 |
| 5 | 79448 | 79296 | 152 | 15,4 |
| 6 | 79690 | 79538 | 152 | 15,4 |
| 7 | 79755 | 79600 | 155 | 12,4 |
| 8 | 79811 | 79659 | 152 | 15,4 |
| 9 | 79963 | 79750 | 213 | 45,6 |
| 10 | 813 | 662 | 151 | 16,4 |
| Average: | | | 167,4 | 18,08 |

Test6 (Camera to TV wirelessly)

| # | Display | Camera | Delay | Deviation |
|-----------------|---------|--------|--------------|--------------|
| 1 | 42852 | 42700 | 152 | 12,7 |
| 2 | 42548 | 42395 | 153 | 11,7 |
| 3 | 42245 | 42061 | 184 | 19,3 |
| 4 | 41788 | 41606 | 182 | 17,3 |
| 5 | 41486 | 41304 | 182 | 17,3 |
| 6 | 38633 | 38480 | 153 | 11,7 |
| 7 | 43277 | 43124 | 153 | 11,7 |
| 8 | 37632 | 37480 | 152 | 12,7 |
| 9 | 37208 | 37025 | 183 | 18,3 |
| 10 | 29383 | 29230 | 153 | 11,7 |
| Average: | | | 164,7 | 14,44 |

Summary

| Test | Devices and software | Delay | Deviation |
|------|---|-------|-----------|
| 1 | Camera to grabber, using cam-share and read | 190,4 | 11,36 |
| 2 | Camera to grabber, using cam-share and Oculus Distortion | 235,2 | 15,36 |
| 3 | Camera to grabber wirelessly, using cam-share and read | 208,3 | 15,38 |
| 4 | Camera to grabber wirelessly, using cam-share and Oculus Distortion | 204,9 | 16,32 |
| 5 | Camera to TV | 167,4 | 18,08 |
| 6 | Camera to TV wirelessly | 164,7 | 14,44 |

II. Quadcopter Speed Test

Test1 Still Standing Start (time to travel 20 meters)

| # | Time[s] |
|---------------|------------|
| 1 | 2,6 |
| 2 | 3,1 |
| 3 | 2,5 |
| 4 | 2,4 |
| | Speed[m/s] |
| Min Speed | 6,45 |
| Max Speed | 8,33 |
| Average Speed | 7,55 |
| Median Speed | 8,16 |

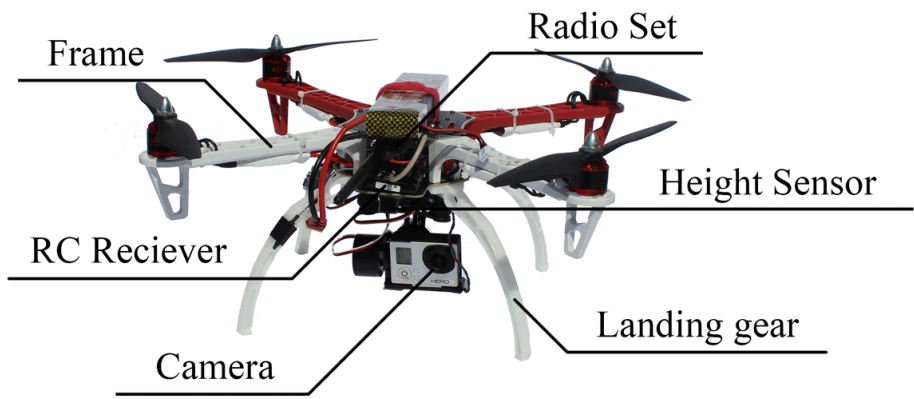
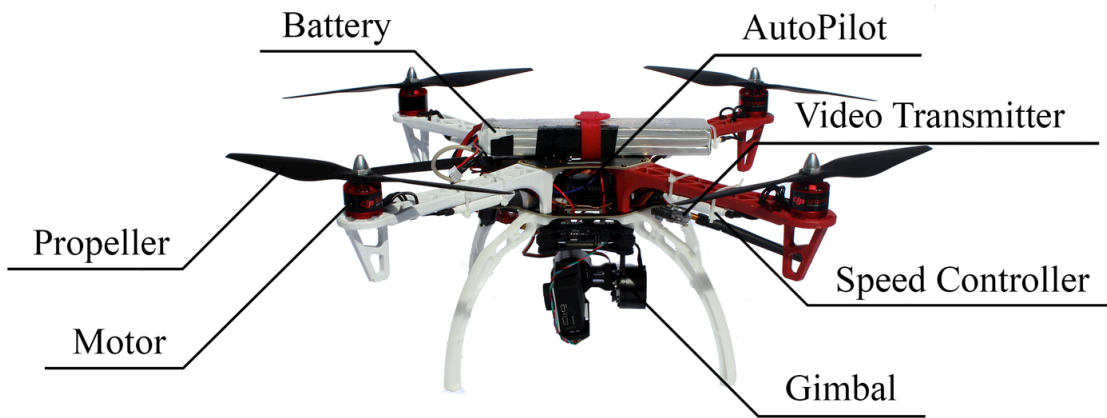
Test2 Flying start (time to travel 20 meters)

| # | Time[s] |
|---------------|------------|
| 1 | 1,2 |
| 2 | 1,5 |
| 3 | 1,2 |
| 4 | - |
| | Speed[m/s] |
| Min Speed | 13,33 |
| Max Speed | 16,67 |
| Average Speed | 15,38 |
| Median Speed | 16,67 |

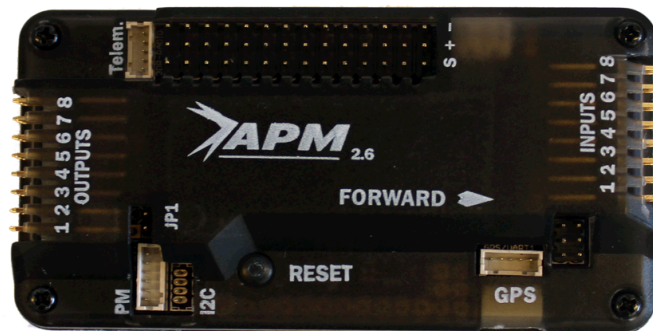
Appendix C - Hardware

I. Quadcopter

| | |
|--------------------------------|---|
| IMU AutoPilot | ArduPilot Mega (APM 2.6) |
| Frame | Flamewheel F450 |
| Landing Gear | DJI F450 Flamewheel Landing Gear |
| Motors | 4x DJI 2212 Brushless 920 kV motor |
| Speed Controllers | 4x DJI 30A Opto ESC |
| Height Sensor | Maxbotix HRLV-EZ4 |
| Radio Set | 3DR Radio Set 433 Mhz |
| Remote controller and reciever | WFly WFT07 2.4 GHz 7-kanaler, incl receiver |
| Gimbal | DJI Phantom Brushless Gimbal w/ Simple BGC |
| Camera | GoPro Hero 3 White |
| Video Transmitter/Reciever | AH-5.8 GHz 200 mW |
| Battery | Gravity 5500 mAh 3S 30-40 C LiPo |
| Charger | EV-Peak V6AC 50 W Multi charger 230 V/12 V |



AutoPilot - ArduPilot Mega (APM) 2.6



II. Oculus Rift



III. Remote Controller



IV. Radio Receiver



V. Charger

