CHALMERS

Immersive Third Person View

Automatic User Tracking with Live Video from an Unmanned Aerial Vehicle

Bachelor of Science Thesis in Computer Science and Engineering

Martin Dahlgren, Jacob Gideflod, Joakim Milleson, Christoffer Palmberg, Filip Saltvik, Christopher Åkersten

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2016

Bachelor of science Thesis

Immersive Third Person View

Automatic User Tracking with Live Video from an Unmanned Aerial Vehicle

Martin Dahlgren
Jacob Gideflod
Joakim Milleson
Christoffer Palmberg
Filip Saltvik
Christopher Åkersten

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
University of Gothenburg

Göteborg, Sweden 2016

Immersive Third Person View

Automatic User Tracking with Live Video from an Unmanned Aerial Vehicle

MARTIN DAHLGREN
JACOB GIDEFLOD
JOAKIM MILLESON
CHRISTOFFER PALMBERG
FILIP SALTVIK
CHRISTOPHER ÅKERSTEN

Examiner: Arne Linde

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96, Göteborg
Sweden
Phone: +46 (0)31-772 10 00

Cover: An illustration of a camera equipped UAV tracking a user.

Department of Computer Science and Engineering
Göteborg 2016

Immersive Third Person View

Automatic User Tracking with Live Video from an Unmanned Aerial Vehicle

Martin Dahlgren
Jacob Gideflod
Joakim Milleson
Christoffer Palmberg
Filip Saltvik
Christopher Åkersten

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Bachelor of science Thesis

Abstract

Third Person View (TPV) is a video game camera perspective where the game character is seen from behind. Due to decreased prices of Unmanned Aerial Vehicle (UAV) technology and modern Head Mounted Displays (HMD) it has become more available to create a system were a user sees themselves from behind. This thesis describes the process of designing a system for creating the TPV with low latency video streaming and short range tacking for automatic flight.
A quadcopter is designed to track and follow the user for achieving automatic camera positioning. Location tracking is implemented with triangulation by ultrasonic sound. Following user rotation is implemented by comparing magnetometers on the user and the quadcopter. From a camera on the quadcopter live video is transmitted by Wi-Fi to a laptop connected to an Oculus Rift Development Kit 2 (DK2) which presents the video feed.
The highest emphasis has been on achieving accurate tracking but also on rapid quadcopter regulation using PID-controllers for maintaining the view. The resulting system works for tracking and reacting to a moving user, however during fast user rotations or movements the tracking is lost, due to small angle of the ultrasonic receivers. For controlling and observing the system during flights and tests, a graphical tool was developed as well as software for tracking and regulation. These systems implemented shows promising results and is an area worth of future study.

Keywords: Third Person View, Quadcopter, Ultrasonic, User Tracking, Automatic, Low Latency, Video Streaming.

Sammandrag (in Swedish)

Tredjepersonsvy är ett videospelsperspektiv där spelkaraktären ses bakifrån. Tack vare minskande priser på drönare och huvudmonterade displayer är det numera mer tillgängligt att se sig själv i tredjepersonsvy. Detta kandidatarbete beskriver hur processen gick till för att designa ett system som ger användaren en tredjepersonsvy av sig själv, med hjälp av videoströmmande i låg latens och automatisk följning av en flygande drönare.
För att få till automatisk kamerapositionering designas en drönare som spårar och följer en användare. Lokalisering av användaren löses med hjälp av triangulation där ultraljud används som signaler. För att följa användarens rotation jämförs magnetometer på användaren med quadcopterns. Från en kamera monterad på quadcoptern skickas direktsänd video via Wi-Fi, till en laptop som strömmar videon på Oculus Rift Development Kit 2.
Främsta fokus har legat på precis lokalisering av användaren, men också på snabb reglering av quadcoptern för att bibehålla rätt perspektiv. Resulterande system fungerar för automatisk följning och reaktion på olika rörelsemönster. Dock är systemet känsligt för snabba rörelser då ultraljudssensorerna tappar täckningen. För att observera och kontrollera systemet under testflygning togs ett grafiskt gränssnitt fram, tillsammans med mjukvara för följning och reglering. De olika systemen visar lovande resultat och bör följas upp i framtiden.

Sökord: Tredjepersonsvy, Quadrocopter, Ultraljud, Användarspårning, Automatisk, Låg Latens, Videoströmning

Acknowledgements

We would like to thank Roger Johansson for his tuition during this project. We would also like to thank Arne Linde for his help with choosing components, general feedback and the opportunity to meet high school students. A thank you to Petter Falkman for the discussion about autonomous tracking, Jonas Fredriksson for a discussion about analogue circuits and Lars Norén for his helpfulness in acquiring components. Lastly we would like to thank David Frisk for the latex-template.

Vocabulary

API	Application Programming Interface, a set of routines or protocols for using a software library
APM	Ardupilot Mega, the flight controller
Arduino	A microcontroller
DK2	Oculus Rift Development kit 2
FPV	First Person View
GPS	Global Positioning System
GUI	Graphical User Interface
HMD	Head Mounted Display
Latency	Delay from input into a system to desired outcome
Oculus Rift	A Head Mounted Display commonly used for Virtual Reality video games
PiApp	Our software running on the quadcopter microcomputer
PID controller	A feedback control algorithm to regulate an automatic system from a measured variable
PWM	Pulse Width Modulation
Quadcopter	Much like a helicopter, it uses rotors to fly, but has 4 rotors turned downwards instead of one downward and one on the tail
Raspberry pi	A single board computer
RC	Radio Controlled
RPM	Revolutions Per Minute

Serial Communication	A form of wired communication where one bit is transferred at a time
SDK	Software Development Kit
TCC	Third Person View Control Centre, our software for system control and monitoring
TPV	Third Person View
UAV	Unmanned Aerial Vehicle
Ultrasonic	Sound with higher frequency than 20kHz which humans cannot hear
VR	Virtual Reality
Wi-Fi	A IEEE collection of standards for radio communication

Contents

List of Figures

1

Introduction

Third person view (TPV) is a popular camera position in the world of computer/video games and can be found in many popular titles like “Grand Theft Auto” and “Watch Dogs”. Unlike the first person view where the camera shows the same perspective as the game character sees, TPV gives an image from behind as per figure 1.1. This enables a better overview of the immediate surroundings, but decreases the sense of immersion.

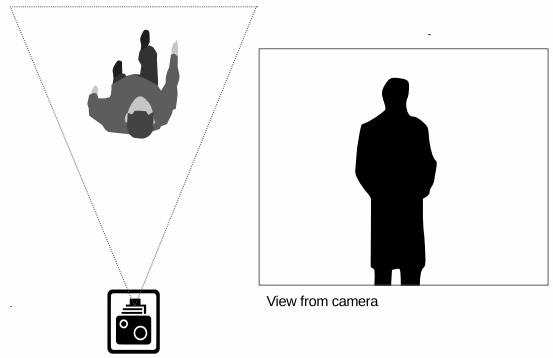

Figure 1.1: Illustration of third person view. The captured video (right) shows person from behind.

This project aims to create a TPV experience in real life. A way to achieve this is to have a camera hovering in the air behind the user. An Unmanned Aerial Vehicle (UAV) such as a quadcopter can be used to achieve this. It would have to navigate at a constant distance and height behind the user and capture video in the same direction the user looks. The captured video can then be live streamed to a head mounted display (HMD) to create the immersive TPV. The past years’ fast technology development, especially in the area of UAVs, virtual reality HMDs and easily available wireless data transfer solutions makes this project possible.

This project is not about solving a particular problem, but rather explores a new area of research "Immersive Third Person View". A TPV does however have some theoretical applications. It would make it possible for game designers to create games where the user have to physically activate himself and thereby break the stationary game culture. Aside from the gaming world a TPV can be useful in technically advanced sports. Instead of recording the athlete on video and then analyse the result after the training session, the athlete can be provided with direct feedback. The result of the project could also potentially be used in other applications not directly related to creating an immersive TPV, by for example only using specific parts of the solution.

1.1 Purpose

The purpose of this project is to develop a solution for a user to get a TPV of themselves in real-time. The TPV will be from behind like in a video game using the third person camera view. This involves the development of an automatic user tracking system which controls a camera equipped UAV as well as a method for video capture, transfer and presentation. All parts of the system should be fast enough to give the user a smooth and responsive experience for an immersive TPV.

1.2 Related Work

There are three bachelor theses conducted at Chalmers University of Technology with the same purpose; J. Allander et al. [1], A. Aronsson et al. [2] and A. Lindgren et al. [3].

In the thesis written by J. Allander et al. it is mentioned that the calculations of the control signal to regulate the unmanned aerial vehicle (UAV) is done at a computer base station. The calculations are done on measurements of height and distance between user and the quadcopter as well as the UAV's position. The signals are sent to the UAV through a communication protocol named MAVLink by the interface MAVProxy.

A. Aronsson et al. describes how image processing was used to track the user. The problem was divided into three parts: centring the camera on the user, calculating distance between the quadcopter and the user and calculating the angle to the user. For centring, the user was detected in the video feed image and the quadcopter was regulated to compensate for deviations. For the distance the size of the target object in the image was measured and related to how large it should be the distance was calculated. The angle to the user was measured by comparing the compass in the user's Oculus Rift and a compass on the quadcopter.

A. Lindgren et al. discussed using three colourful balls in a triangular pattern on the user back to easy be identified by image processing. The balls were tracked and from their spanned triangle's size and skew distance, angle and location was determined.

1.3 Problem Definition

The selected approach for this project is to use a UAV, more specifically a quad-copter, along with the head mounted display Oculus Rift Development Kit 2. This leads to several challenges that need to be addressed. The problem space has been divided into the subproblems depicted in figure 1.2.

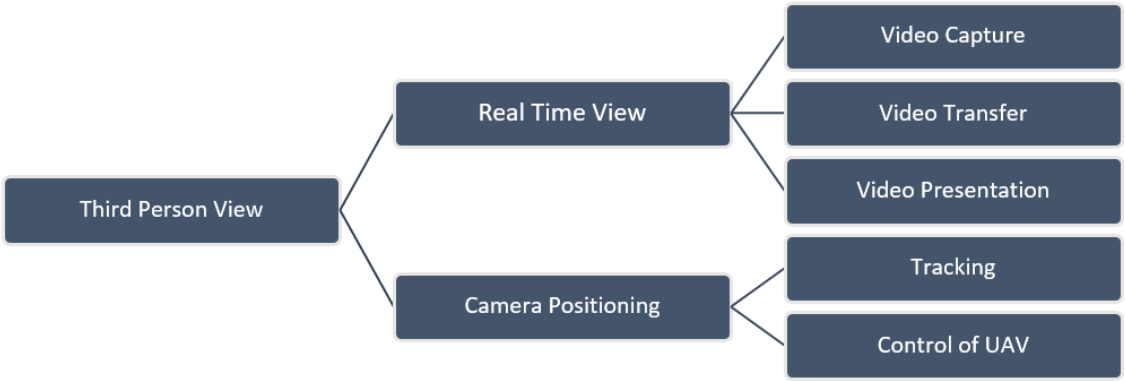

Figure 1.2: A depiction of the problem space, dividing the main goal "Third Person View" to several smaller subproblems.

1.3.1 Real Time View

The real time view is responsible for presenting the user with live video. One of the problems with the real time view is to achieve high image quality while keeping very low latency. This puts hard requirements on all parts of the real time view; video capture, video transfer and the presentation of the video. In order for the real time view to feel instantaneous the entire process from image capture to the user seeing the image cannot take more than 100 ms [4]. The guidelines set up for the chosen head mounted display provides even stricter recommendations of less than 20ms latency [5]. Further the image quality and refresh rate of the video display must be high [6] which further increase the amount of data to be transferred in a short time.

1.3.2 Camera Positioning

To achieve an immersive third person view experience accurate camera positioning is of major importance. The goal is to keep the camera centred from behind the user at a constant distance. This is to be done automatically such that the UAV moves with the user and constantly repositions the camera to centre the user. The objective is to make sure the camera is capturing video in the same direction as the user is looking, while maintaining the user in focus. Thus when the user rotates the UAV must move in an arc around the user, as described in figure 1.3.

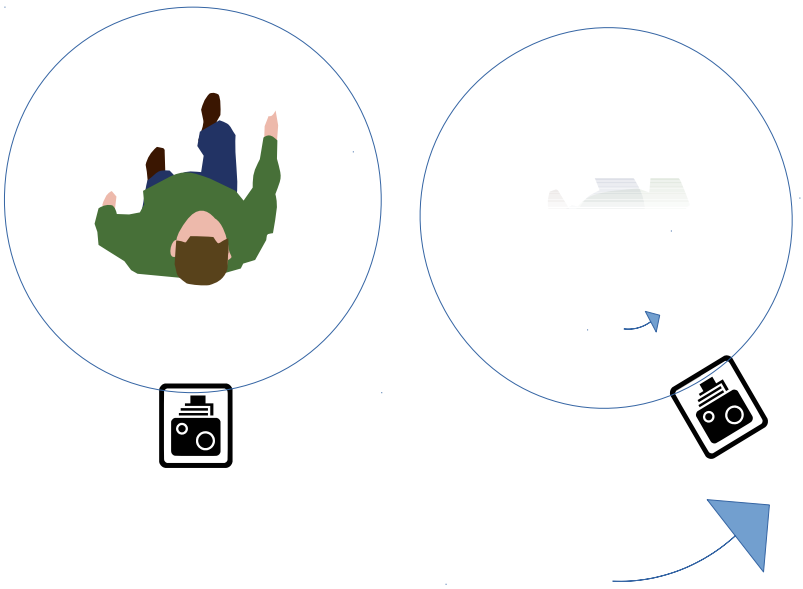

Figure 1.3: Rotation around user

In order to achieve this behaviour the orientation and location need to be constantly measured and the position regulated. There is a state where no regulation will be performed, and that is the "steady state". In this state the UAV is located correctly at a set distance and height, whilst keeping the orientation equal to that of the user. When a measurement is different from that of the steady state it should result in a regulation of the UAV in order to counter this difference and return to the steady state. The parameters must be monitored and regulated fast enough to provide a safe and satisfying user experience.

1.4 Scope

The approach for the camera positioning will be strictly limited to an UAV, as per the request if the institution. Further, the developed system will not take its environment (i.e. obstacles) into consideration and will only focus on following the user in an open area. This decision was made because of the amount of work to detect and handle surrounding objects.

The quadcopter will be limited to tracking a user. Hence it will not be able to find the user, meaning that it will at startup be placed in such a position where it can immediately detects the user. Further if the quadcopter somehow lose track of the user it will not be designed in such a way that it can automatically find the user again.

1.5 Methods

This section presents the method for development and implementation of the prototype. As stated this project deals with the development of a real time third person view system. The method of development was to initially divide the project into subproblems as described in the problem definition (section 1.3). The subproblems were composed into three problem areas, which were each assigned to a responsible team to enable simultaneous development of the different parts. These three areas were

- Real Time View (video capture, transmission and presentation)
- Tracking of user location and orientation
- Quadcopter operations; including assembly, manoeuvring and automatic control.

These three areas were developed using an iterative approach, where different approaches to solve each subproblem were implemented, tested, reviewed and adjusted as necessary. Ideas for solutions were brought up from literature studies of earlier Third person view theses and other sources. Subsystems were developed and at this stage separately tested in a lab setting. The tracking system for example was tested by moving the sensor equipped quadcopter manually and reviewing the result, parts of the quadcopter system was tested during manual flights and so on.

Continuous collaboration was done between the working groups to enable the developed subsystems to share mutual support systems such as computation unit on the quadcopter and communication links. When the separate parts reached their completion they were incorporated into the final prototype. Here shortcomings of the individual systems were identified which were then further developed in the lab; either by improving the solution or trying alternatives.

Software was developed for handling the incorporation of the different parts and to provide tools for testing, observation and logging.

An alternative method would have been to create software and models for system simulations rather than using the trial- and error approach for prototype building. The outlined method was chosen due to time- and resource constraints for as well as lack of experience of constructing realistic simulation models.

1.6 Report Outline

In this thesis there are the following chapters: Introduction, Theory, Development and Implementation, System Tests and Performance and Discussion. Theory covers the technical knowledge needed to fully appreciate the Development and Implementation section. Development and Implementation describes the process of developing and implementing the system as well as the design choices made. It is further divided into describing the different modules, the control software and lastly final assembly. The system tests and performance section outlines the performance of the prototype, how well it works and provides technical data from tests. The discussion concludes what was good, what could have been done different and thoughts about future work building on this thesis.

2

Theory

The development and implementation section of the report requires knowledge about the technologies used and theory surrounding terms, systems and components. This section seeks to provide the technical knowledge needed to fully appreciate the development and implementation section.

2.1 Digital Video Encoding

For a digital video representation there are various possible encoding standards suitable for different applications. There are both lossless and lossy encodings used for compressing video. Two popular lossy encoding standards are MJPEG, where each frame is encoded according to the JPEG standard, and h264, also called AVC.

Compressing the video data greatly decreases the size of the data. Depending on the type of compression method used it can however lead to buffering requirements. h264 encodes a single image frame as either an I- a P- or a B-frame. All types are usually used in a video stream, but mainly B frames are interesting with regards to latency. B- (Bi-directive) frames are video frames that can not directly be rendered as they do not contain a full image, they depend on frames later in the stream and leads to buffering requirements. For low latency streaming B-frames should be deactivated in the encoding.

Different video encodings requires different bit rate when transferring and viewing the video. Bitrate is the amount of data that needs to be processed per second. Using uncompressed video data requires very high bitrates. A standard full HD (1080 by 1920 pixels) video stream with 60 frames per second and 8 bits per colour per pixel would for example require a 3 Gbit/s bitrate. This is also the highest possible resolution and frame rate the DK2 can use. Using a compressed video stream with the encoding h264 for the DK2 takes 20 Mbit/s and lastly MJPEG would take 100Mbit/s [7]. Given enough bandwidth uncompressed video encoding can lead to very low latency as an uncompressed image from the video camera can be directly sent without any time consuming processing.

Lossless video compression has the same drawback as lossy compression that it requires time consuming compression algorithms but can theoretically lead to higher image quality than lossy encoding as it will be possible to recreate all captured data.

2.2 Head Mount Display Optical Distortions

The proposed display to show the real time view is the head mounted display Oculus Rift Development Kit 2 (DK2). In the DK2 a flat screen is placed a few centimetres in front of the user's eyes. For the eyes to be able to focus on the screen lenses are placed in front of the screen. These lenses have the drawback that the picture will get distorted. The two kinds of distortions are Barrel distortions and Chromatic aberrations. The barrel distortions will make the magnification level differ in the image, objects closer to the middle will look more magnified than objects in the fringe as illustrated in Figure 2.1.

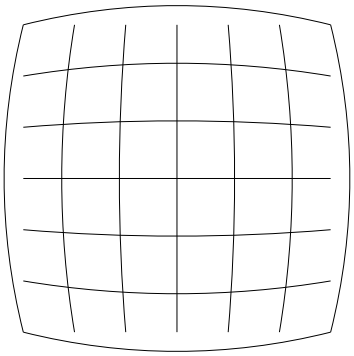

Figure 2.1: Illustration of Barrel Distortion on square grid

The Chromatic Aberrations makes the colours in the picture smeared. It is due to the lenses bending light of different wavelengths by different amounts as illustrated in Figure 2.2. There are video filters limiting the effect of both Barrel distortions and Chromatic Aberrations at varying success [8].

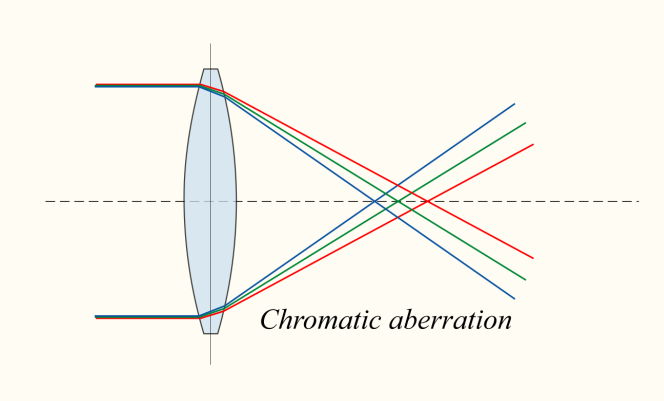

Figure 2.2: Illustration of Chromatic Aberration. By DrBob at the English language Wikipedia under CC BY-SA 3.0 license

2.3 Control of a Quadcopter

To be able to make the quadcopter follow the user automatically it is essential to be familiar with its possible movements. Since a quadcopter is inherently unstable [9] it needs feedback when controlling the motors to making it stable. That is what a flight controller is doing. Depending on the input, velocity, rotation and other data it sets individual RPM (speed) values for each motor. For this project the flight controller Ardupilot Mega is available, in which there are four user settable parameters to control the movement of the quadcopter. These are the throttle parameter which controls the climb rate, the pitch controls the forward tilt, the roll controls the sideways tilt and the yaw controls the rotation around the quadcopter's own axis. The available movements are shown in Figure 2.3.

The flight controller can achieve the possible movements due to the quadcopter rotors spinning at different directions. Thanks to advanced algorithms the quadcopter can move in all directions just by varying the rotor speeds. How this is implemented is not in the scope of this project and will not be mentioned further.

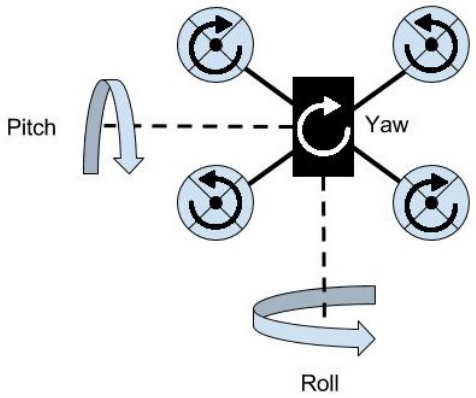

Figure 2.3: The movements of a quadcopter.

2.4 PID

For translating sensor readings to appropriate control commands a control algorithm must be used. Some sort of controller for regulating autonomous systems is common [10] and a common controller is the PID-controller. It is used to set appropriate control commands to a system from measurements on the system, e.g. setting the upwards throttle on a quadcopter from readings of its height. It provides both speed and stabilization of the system. A PID-controller has three variable parameters, gain coefficients for the three terms proportional, integral and derivative term respectively.

Proportional P-effect: The proportional term relates to how far away a reading is from its target value, the further the stronger control signals are generated. A high proportional gain will increase the speed of the system and improve compensation of process interference as well as a rise in signal activity. However, the system runs a higher risk of instabilities.

Integrating I-effect: The integrating parameter further increases the speed of the system. It relates to stronger control signals from how long time the system is away from its target state. In the case of the quadcopter this gives increasing control signals as long as it is not in place i.e when the quadcopter has yet to reach its position. Just as the proportional parameter the integrating parameter also contributes to instabilities. The I-part is also required as a P-controller for many systems give a constant error, that is one can never reach the target steady state with just a P-controller as it generates no control signals for zero error.

Derivative D-effect: The derivative part reacts to fast changes in the tracked state. It adds control signals proportional to the derivative of the measured value thus gives a fast response to disturbances. Most systems manages with a PI-controller but when speed is of the essence the D-parameter also makes it possible to have high proportional and integral parameters without inducing oscillations[10].

3

Development and Implementation

This section describes the process of developing and implementing the system. It seeks to first describe the development process of the different modules that the system was divided into, then the implementation of the final prototype.

3.1 Real Time View

The purpose of the Real Time View module is to achieve low latency, high quality video streaming from the quadcopter to a HMD. Our choice of method for video capture, transfer and presentation will affect performance both regarding latency and video quality. Thankfully it is not the first time this has been done. Previous works at Chalmers University where J. Allander et al. [1] and A. Aronsson et al. [2] have attempted to solve the same problem, and gave inspiration for design and development. This chapter describes the implementation, development and design choices made to create the real time view module.

3.2 Video Format

In order to achieve low latency in the real time view system the video representation is important. Video data can be represented in two different formats, digital and analogue. Today there exists both digital and analogue solutions for all steps required in the system; video capturing, transmission and presentation. One of the deciding factors for the video format where the HMD. HMDs in general support either digital or analogue input. Requirements for the HMD was high frame rate and resolution for great video quality. There was a digital HMD the DK2 available. There are both digital and analouge HMDs on the market that performs better, however no alternative where found that out peprformes the DK2 while being within our budget. Hence the choiche of HMD was the DK2. The DK2 only has a digital video input, which means that the video data has to be in a digital format at the presentation step.

Hence the option stands between using either a completely digital system or converting from an analogue signal to a digital format somewhere in the process. A. Aronsson et al. have used the latter approach and got high latency levels from the video conversion [2]. They argued that a higher quality analogue frame grabber would be a solution to lower the latency. In an interview with Roger Johansson, Feb. 2016, he stated that well performing analogue to digital converters are in general very expensive. No low latency frame grabbers was found in the market that could fit into the project budget. J. Allander et al. used a complete digital approach with good results [1].

3.2.1 Video Capture

It was decided to use a similar hardware set-up for the video capture as in the work by J. Allander et al. [1] For capturing video the single board computer Raspberry Pi model 2B was used, hence after referred to as the Pi, along with the corresponding camera component developed specifically for the Pi, the Pi camera module V1. The Pi is relatively cheap and has hardware accelerated h264 encoding capabilities [11]. The camera component offers the performance required offering high enough resolution and framerate suitable for the DK2 [12].

J. Allander et al. [1] used the raspivid module, so that would be a natural choice. However the other alternatives where also interesting for various reasons. There was an interest in exploring the option of capturing uncompressed video. The picamera drivers, compared to the alternatives has that option of supporting uncompressed video. Picamera was discarded however since even the authors of the drivers were unable to reach video capturing above a 15fps rate [13] and no implementation were found where a satisfying result could be reached. Instead the UV4L (User space Video 4 Linux) platform was tested and compared to Raspivid. The UV4L offers a complete video streaming solution with camera capture drivers and a streaming server. Raspivid is simply the drivers for capturing a video stream possibly using pi's h.264 hardware compression. The UV4L platform was compared with Raspivid combined with GStreamer for video transfer introduced in the next section.

3.2.2 Video Transfer

Transmission of the data was done over radio with the well established Wi-Fi protocol. Options to radio transmission were explored such as an optical system or audio system, however no viable alternative to radio transmission was found. Alternatives to the Wi-Fi protocol were other standards such as Bluetooth, however no competitive alternatives to Wi-Fi was found that could match the speed and simple setup. For Wi-Fi transmitters and receivers the TP-Link TL-WN722N Wi-Fi dongle was used on both ends of the connection. They use the IEEE 802.11ac standard and offers a theoretical speed up to 150Mbps [14]. The receiver was a personal computer connected to the DK2.

There was an option to use Wi-Fi in ad-hoc mode or infrastructure mode i.e. with or without a central base station. The infrastructure mode was used with the Pi acting as a router. A wireless network was set up with the Pi as an access point with the software hostapd. Maximum throughput was tested with iperf (a standard network testing tool) and showed a maximum of 28Mb/s with TCP and 46Mb/s with UDP. The reason for the low speeds was uncertain. One plausible theory was that the USB link on the Pi throttles the Wi-Fi link performance. Another was disturbances, however changing environment to one with less radio interference did not improve the results. The low speeds made it clear that high compression levels would be needed.

The encoding of the video for the video compression was made with the *Raspivid* utility where various encoding and compression options have been tested for optimal performance. The main option for the encoding offered by Raspivid is h264. J. Allander et al. [1] got the best performance with the MJPEG encoding. There where however no success in repeating the feat due to lack of documentation of how J. Allander et al. [1] did it and the lack of software supporting handling MJPEG streams and pi's MJPEG hardware acceleration. GStreamer is an open source multimedia framework where various modules can be linked to process video streams in various ways. There were no competitive alternatives found for handling the video streaming. The UV4L platform can handle both h264 and MJPEG encoding, however it did not perform well. Hence GStreamer was used together with the h264 encoded stream from Raspivid.

Raspivid was set to output a h264 stream that was piped to the software *GStreamer 1.0*. There were few/no research studies available for the optimal GStreamer pipeline for processing the video stream. Instead ideas from different unscientific sources where similar attempts had been made were tested. Experiments with various solutions for the pipeline were made and in the end the one with the best performance for the application was chosen. In this case a GStreamer pipeline was created using modules for packing the h264 stream in RTP packets and transmitting it by UDP over Wi-Fi to the receiver. The full command to start the video transfer is listed in Appendix B.4.

3.2.3 Video Presentation

The video presentation was done on the DK2 which is connected to a laptop. Tests were performed to use the Pi instead of the laptop but they were for unknown reasons unsuccessful. There is no official support for the Pi which is a part of the complication, therefore a regular laptop running Windows 7 was chosen. When presenting video on the DK2 there is a need to compensate for Barrel distortion and Chromatic Abbreviation correction. The DK2 software development kit, hence after refereed to as the SDK, has libraries for applying both Barrel distortion and Chromatic Abbreviation correction [15]. However, to utilize these filters one has to process the video through OpenGL or Direct3D, implementing this seemed like unnecessary work for 2D-video streaming. It might be possible to make or find filters correcting the distortions from the HMD that easily integrates into our video pipeline. However, due to the time constraints these options where not explored and left as further research. It is also worth to mention that J. Allander et al. [1] previously succeed with correcting for Barrel distortion and Chromatic Abbrevation using the SDK. It did however add severely to the latency of their system going from about 50ms latency to about 195ms latency [1, p. 29].

The laptop was connected over WiFi to the wireless network set up by the Pi. For receiving the video stream GStreamer was used. The GStreamer pipeline used to show the image is mostly the reversed send pipeline. A UDP-source is piped to a rtp depayer via a h264 decoder to an automatic video sink. When launching the pipeline with gst-launch-1.0 a window is opened that shows the image stream. The Oculus works as a HDMI-screen. The window with the video is therefore simply placed on the DK2 screen in fullscreen. The Oculus handles the duplication of the image to give the same image to both eyes [15]. The performance of the real time view module is presented under the chapter 4.

3.3 Tracking

The main goal of the tracking module is to continuously track a moving user's location and rotation relative to the quadcopter, in order to provide input to the quadcopter regulation module. The tracking should be accurate and fast enough to make following of the user feel smooth. In addition to user tracking, the continuous height measurement will also be described in this section.

For creating the third person view the camera should always have the user centred and make sure that the camera captures video from behind the user i.e. in the same direction as the user is facing. The camera should also be at a constant distance from the user as described in section 1.3. This fact is important for how to represent the tracking data as there is only a need to track the relative location of the user to the quadcopter and not to an absolute geographical location. Henceforth the situation where the third person view alignment is achieved will be called the steady state.

A relative location can be represented in different ways. One way is by a combination of the direction towards the location and the distance to the location. Since the user is moving on the ground plane there is only the need for a single angle to determine the direction, the angle from the camera centreline. The combination of the angle and the distance gives a user location in the quadcopter's coordinate system, but there is one thing more required by the tracking.

By knowing the location of the user the quadcopter can determine if the user is on the centreline or otherwise knowing where to move to make it so. As stated the user should also be viewed from behind. This further requires tracking of the user rotation, that is where the user is facing compared to where the quadcopter front is facing. This rotation can also be represented by a single angle if one assume the user is always looking alongside the horizontal plane, neglecting upwards or downwards tilt.

This gives three numerical variables to track:

- Distance to user
- Horizontal angle from the quadcopter centreline
- Difference in user and quadcopter angle of rotations

Ideally when the third person view is achieved all these variables should be constant. Therefore the primary goal of the tracking module is to make the tracking work close to the steady state with the assumption that the quadcoper regulation module will make the user stay in the steady state.

3.3.1 Possible Tracking Methods

During early project phases many solutions for a tracking system were evaluated. A straightforward alternative was video analysis of the available video stream as this had been implemented by previous instances of the third person view project [1]–[3].

With video object recognition the size and location of the tracked objects in the video stream are calculated. By knowing the sizes and shapes of the objects the distance and real location of the tracked objects are determined. By tracking a non-symmetrical object or several objects in a constellation the angle towards the object can also be determined. To find objects visually a lot of heavy calculations has to be done as per A. Lindgren et al. [3], which is non-optimal given the requirement of rapid tracking capabilities. If these calculations are done on hardware not directly connected to the camera latency from the video transportation will also add to the tracking latency.

In the previous reports [1]–[3] there are mentions that latency was an issue. With this in mind, and after discussions with the supervisor it was decided to try out something different; some type of transponder or beaconing system. The prospect was that this could provide a faster system, avoiding the heavy computations thus enabling increased sensor data updates.

3.3.2 Distance Measurements Using Waves

Travelling waves, both electromagnetic and acoustic waves can be used to measure distance. This can be done by a method called Sonar for acoustic waves (or RADAR for EM-waves in the Radio frequency range), a pulse is generated which travels by a constant velocity towards the tracked object. The pulse bounces on the object which means that after some time a pulse response can be sensed. If the pulse transmitter and generator are close to each other the waves travels two times the distance which can be calculated as:

$$\text{distance} = \frac{\text{travel time} \cdot \text{travel velocity}}{2} \tag{3.1}$$

For our application both the transmitter and receiver would have been placed on the quadcopter frame and directed towards the tracked user. There are possibilities to buy pre-made integrated modules with this range finding ability by ultrasonic sonar or Infra-red light. Another option is to develop own modules from available transducers.

An alternative to using echoes or reflections for measuring distances is to have separate transmitter and receivers. A pulse is transmitted from one side of the measured length. A receiver is placed on the opposite side and the time from that the pulse is transmitted and received is proportional to the distance. The principle is the same as for the previous case, but now the wave is not reflected and the wave only travels a single distance. With both alternatives it is also possible to determine angle towards the user as will follow. An advantage of using sonar is that the absolute distance to the user can directly be calculated from the time difference between sending a pulse and receiving it. For measuring distance with separate transmitters and receivers the receiving system need to know the time the pulse was sent. The drawback with sonar is that much of the pulse energy will be lost in the reflection, much due to the user not being a flat hard surface. Because of this the module would have to be perfectly directed at the user so that sufficient energy for pulse detection is received.

Direction towards the User

Finding out the direction towards the user relative the quadcopter's centreline was based on a simple idea: There are two distance measurements done between different points on the quadcopter. This can be done by two separated receivers on the quadcopter that get hit from a signal from the same source. These receivers are at a known distance, and by comparing the reported distance the centreline angle deviation can be calculated. The idea is presented in Figure 3.1. In a conference paper by O. Wijk et al. [16] they use a similar technique but for a different purpose. The technique works well for angels between ±90, but for larger angles it will be

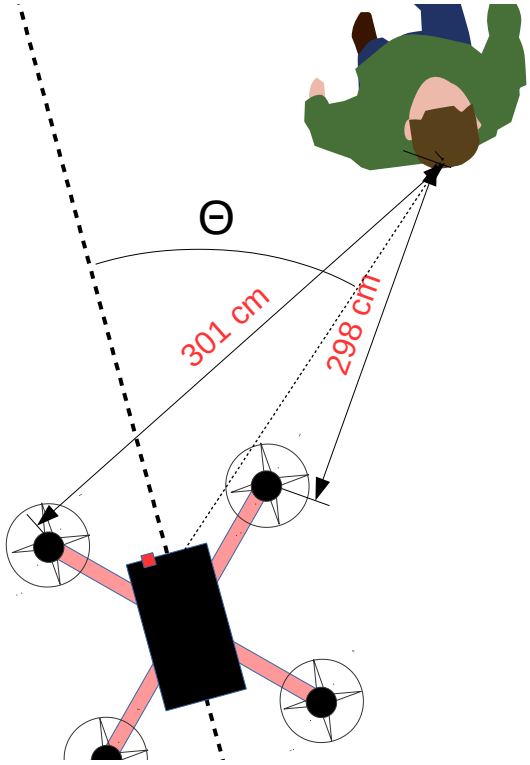

Figure 3.1: This figure shows the angular offset from the the quadcopter's centreline, and the distance measured from two sensors

impossible to determine whether the signal was sent from the front or the back of the quadcopter. If θ in Figure 3.1 is more than 90°, e.g. $\theta = 90° + 10$ it will yield the same result as $\theta = 90° - 10 = 80°$. One way to distinguish the two cases is by adding a third receiver, not placed in line with the other. This setup leads to a certain determination of the angle.

When the quadcopter is at the steady state it is located 3 meters directly behind the users head. A well working regulation of the quadcopter keeps the location close to the steady state, which ensures that $|\theta|$ is less than 90°. Therefore the receivers and transmitter will in practice always be directed towards the user or the quadcopter and the third sensor becomes redundant. As depicted in Figure 3.1 it was decided to put two receivers on the quadcopter frame.

Next choice was to determine what type of signal was to be used. Various alternatives were considered, such as laser distance measurement systems, ultrasonic systems, radio systems and Infra-Red light systems. Laser ranging can provide a very exact distance measurement[17], but due to being very expensive this was discarded. IR-light was also discussed, but the main idea to measure the time difference would be hard using light, due to light being so fast. If the sensors would be placed 30 cm apart, a reasonable distance due to quadcopter size, and being hit straight from the side, it would take 1 nano second between the pulse reaching each sensor. A computer would need to check if each receive has been hit at least at a 1 GHz frequency just to register a difference in receive time. The Raspberry Pi which would be used on the quadcopter has a clock frequency of 900 MHz, which is too slow. Given this the choice fell on ultrasonic sound [18], sending and receiving pulses of it.

Ultrasonic Modules

With a speed of 343 m/s at normal temperature and air pressure[19] ultrasonic sound is suitable for the application. Sound is the compression of air, and its spread has a direction. Available ultrasonic elements (transmitters and receivers) are directed, with different beam angles. In order to function as intended the ultrasonic receivers must be directed towards the transmitter and vice versa so they are within the beam angle.

Ultrasonic sonar modules of the models "Parallax Ping)))"[20] and "HC-SR04"[21] were tested. These are compatible with both a Raspberry Pi and an Arduino Uno [22]. The Ping))) does not do anything until provided a digital trigger signal. When triggered, it sends an ultrasonic pulse, and starts to listen for the echo. As separate transmitters and receivers were considered, alternatives were searched for which could continuously listen for pulses without first sending a pulse. Another problem with Ping))) is its narrow beam angle, which is ideal for sonar if the module is perfectly directed but not for tracking of a moving target.

To overcome the shortcomings of the combined transmitter/receiver ultrasonic modules, effort was made to create new modules. Ultrasonic transducers were acquired which had a better transmit angle. However to make use of these, amplifiers had to be designed, these were never used but the design process is documented in section 3.3.6.

The digital sonar module Ping))) was used, they have an inbuilt limitation in that they only listen for an echo of the sent pulse during the time to detect a three meter distance, which might be to short. Further it was desired for one transmitter to send pulses to be sensed by two receivers. Therefore it was decided to go with separate transmitters and receivers. This can be done by using the combined sonar modules if the pulse generation is prevented on the receiver module and vice versa. The receivers will be triggered to send a pulse and wait for a response, but as the pulse generation is prevented the module will just wait for the identical pulse sent from the transmitter module. To read a distance with this setup the transmitter and receivers must be synchronized.

3.3.3 Measuring Distance with Separated Pulse Transmitters and Receivers

As stated before, it is possible to use digital ultrasonic modules. The available modules are combined transmitters and receivers for sonar ranging, but have been modified to either work as transmitters or receivers. For measuring absolute distance and also because the modules are only listening in a short interval after being triggered there must be perfect synchronization between pulse transmit and the listening of an incoming pulse.

Initially the proposed method was to use the Wi-Fi link for synchronizing. Both the transmitter and the receivers were connected to Raspberry Pi:s with Wi-Fi capabilities. A software was created that sent a short message from the receiver for requesting a ultrasonic pulse and then triggering listening for incoming pulses. When the message was received at the transmitter Pi, it sent the ultrasonic pulse. If the message had been received and the pulse sent in a short enough time, this would have given an absolute distance reading.

However, the distance readings did not work because of network jitter. Had the messages been transmitted with a constant latency the delay could have been compensated for, but the message arrived some time after about 8-10 ms. This jitter of 2 ms made it impossible to give an exact distance measurements as it accounts to an error of 70 cm. A potential solution might have been to instead of a simple request of pulse transmit, send a message with specific future time for pulse transmitting. Supposing the clocks was synchronized the transmitter would send the pulse at a time known by the receiver. Since the clocks on the Pi:s were suspected not to be perfectly synchronized a frequency based solution was instead implemented, that do not depend at the Wi-Fi link.

Method of Clock Synchronization

It was decided that the transmitter on the quadcopter should send ultrasonic pulses at a constant interval of 60 ms. An Arduino microcontroller is connected to the transmitter which uses its internal clock to time the pulses.

The quadcopter's receivers sense the pulses, but must know when they were sent to determine the distance. To solve this problem there is a start up phase before flying where the user transmitter and quadcopter is placed at the desired steady state distance (e.g. three meters). When the receivers sense the first pulse the current time is saved in a variable t_f. Since the pulses are transmitted at a constant frequency and should travel a constant distance all the next pulses should be detected at times:

$$t_n = t_f + n \cdot 60 \text{ ms, where } n = 1, 2, 3, 4... \tag{3.2}$$

If pulses are detected at any other times the distance to the user differ from the target steady state. The difference between time of detection and closest time t_n is Δt. By using the simple relationship between time and sound velocity the distance from steady state is calculated as:

$$\Delta distance = \Delta t \cdot \text{speed of sound} \tag{3.3}$$

As the steady state distance was determined during the start up phase the distance measuring in theory should now be complete. For how to actually implement this behaviour with the sensors refer to the source code B.1.

Clock Drifting

A problem was identified after implementation, the distance measuring was drifting. The reason for this must be that the internal clock of the Pi, the Arduino or both was not correct and did not measure the time accurately. To overcome this drifting of the clocks a constant was added called the "clock constant" *cc*. This constant was made to compensate for the drifting, the Pi program assumes that the pulses comes at an interval of not 60 ms according to its internal clock but of 60ms $+ cc$. The constant was adjusted so that the distance measuring worked without any drifting and seemed to have solved the problem.

The next problem was identified the next time the system was tested, the distance measuring drifting had reappeared in spite of the added clock constant. The reason for this is probably that the clock drifting itself is not constant, and changes due to unknown causes, this might be because of for example temperature differences or something else. The clock constant therefore has to be reset at every system use to compensate for the drifting at that occasion. In the future this might be automated and done at the start up phase, but now setting the clock constant is done manually to get a stable distance measurement for each time of flying.

3.3.4 User Rotation

The Oculus Rift Development Kit 2 has built in magnetometers [15]. By measuring the magnetic field these can act as a compass [23]. In order to stay directly behind the user's head, the DK2's compass angle would be compared with the compass angle of the APM i.e. the rotation of the user compared to the rotation of the quadcopter. The goal is that the quadcopter camera should look the same way as the user, as the quadcopter is to be placed behind the user.

This together with the correction of angular displacement should keep the quadcopter in the desired steady state. In practice to read the compass angle from the DK2 a laptop has to be connected to the DK2 by USB. A program was written in C++ to get rotation data with the official Rift software development kit. The SDK provides rotational data round all axis but the one of interest was rotation in the horizontal plane; the yaw angle.

A problem was found when using the Linux version of the SDK as this did not actually make use of the magnetometers. It derived the user rotation from the inbuilt accelerometer. Using accelerometer data to derive location or rotation is prone to drifting values as even small errors of the accelerometer readings will add up over time. This proved to be a real problem as the reported angle drifted even when the DK2 laid still. With a Windows computer another version of the SDK could be used where the yaw angle was derived from sensor fusion of the accelerometer and the magnetometer reading which gave a more stable reading.

Another problem was that the SDK API did not report yaw as a value from compass north as the APM did, but rather from a seemingly random initial value. It should be possible to derive the real north from using the DK2's low level sensor values. As these are not in a form directly usable and would need filtering etc. it was not investigated further. The problem was instead solved by having an initial phase before flying; the DK2 is assumed to be started from the steady state position where the quadcopter's compass angle is at that point the same as the DK2's. The offset between the APM's real compass bearing and the DK2 SDK's reported yaw is saved as a constant. This constant is added in all further calculations so that the reported yaw is converted to angle from magnetic north.

3.3.5 Height Measurement

To maintain a given height the initial thought was to use the quadcopter's built in barometer. After the first flight test it could be seen that the barometer precision was ±1 meter, which was not sufficient for our low altitudes. It was therefore decided to add an ultrasonic sonar facing downwards to measure the height. The ping sensor provides good height measurement (at best ±3mm) up to 4 meters [21], which is sufficient for the application. The drawback is the sensitivity for turbulence and vibrations [24]. Both Parallax Ping))) and HCSR04 was tested during flight. The hypothesis was that Ping))) would be better as it supports a higher refresh rate (a new reading can be started directly after the previous is finished) than HCSR04 which requires a 60ms interval[21]. A higher refresh rate is beneficial for the regulation, however it was seen that the Ping))) was more sensitive to disturbances during flight and provided a lot of errors. To get accurate readings HCSR04 was chosen despite the 60 ms minimum refresh interval (16,7 Hz). When the quadcopter

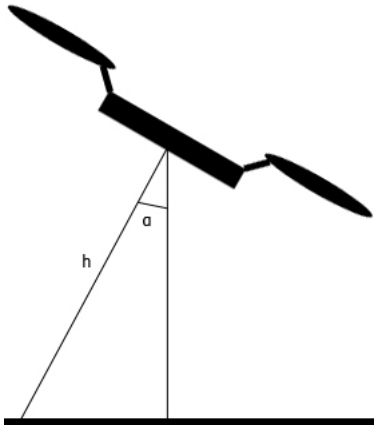

Figure 3.2: Representation of tilt caused by the quadcopter moving

is rolling or pitching, as explained in Figure 2.3 on page 9, it will not be horizontal and the ultrasonic sensor will report a higher height than it should since it will be facing the ground diagonally and not straight downwards as shown in Figure 3.2. Fortunately the APM has an inbuilt gyroscope, the quadcopter inclination is therefore known. The algorithm used to solve the height measurement problem with this is:

$$\frac{h}{\sqrt{\tan^2 \alpha_r + \tan^2 \alpha_p + 1}} \tag{3.4}$$

where h is the measured height, α_r is the angle of rotation from the roll of the quadcopter and α_p is the angle from the pitch of the quadcopter. The derivation of this formula can be found in Appendix B.3.

The readings from the Ping))) sensor are sometimes wrong due to outer circumstances or malfunctioning hardware. To decrease the effect of these corrupt readings a simple filter was implemented. The filter design is based on that a new reading cannot differ too much from the previous one. The height measurement cannot differ more than

$$V * dt \tag{3.5}$$

where V is the average climb rate between the two most recent readings and dt is the time between the readings. The average climb is calculated from the APM accelerometer readings. The new height reading is therefore constrained between

$$h_0 - V * dt <= h <= h_0 + V * dt \tag{3.6}$$

where h_0 is the previous reading.

3.3.6 Analogue Sensor Development

The digital ultrasonic sensors that was available (HC-SR04 and Ping)))) listen with a given algorithm and was directed with a narrow angle, analogue receiving elements was looked into. They should be as sensitive and have as wide beam angle as possible. The receiving sensors produces a signal which is quite low, a few mV at most. In order to read an analogue value an Arduino would be used, the Raspberry Pi only supports digital inputs. Because of the functionality of the Arduino, the value would have to be a few volts, up to maximum of 5V. The Arduino converts the voltage measured into a value between 0 and 1023, where 5V is represented by 1023, and 0V by 0. Therefore an amplifying circuit needs to be made. The circuit needs two main functions to be able to read the signals well; amplification and filtering. The only frequency of interest is the frequency produced by the ultrasonic transmitter i.e. 40kHz. The best type of filter is a band pass filter. It should be designed so that the centre frequency of the filter is the same as the receiver's ideal frequency. The Arduino's analogue input should read 0 continuously until the receiver is hit. Then a higher value should be recorded, and by implementing a software algorithm it is possible to time the hit.

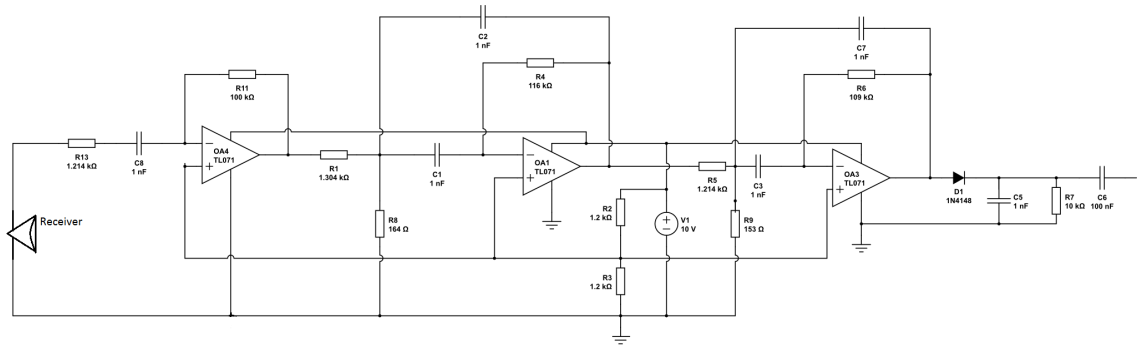

Figure 3.3: A first stage amplification with an active 4th order Butterworth band pass filter. The values were calculated using [27]

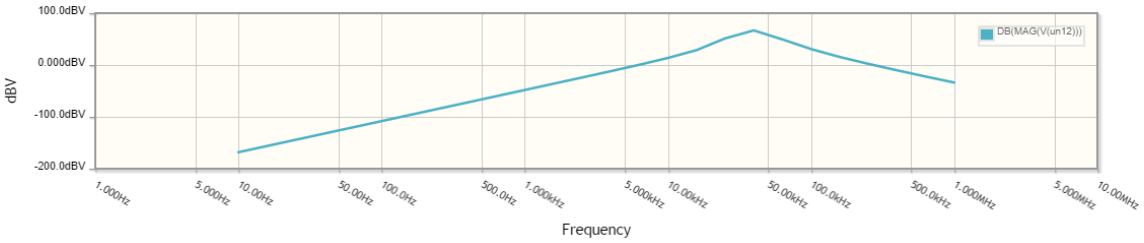

Figure 3.4: Bode diagram showing amplification plot for the circuit in Figure 3.3

There are several different filters with different characteristics. The decided model, seen in Figure 3.3, was based on a Butterworth filter which has a flat pass response [25]. A three stage circuit was implemented. First stage is an ordinary band pass filter amplification, and the second and third combined is an active 4th order Butterworth band pass filter. These should yeild 66.7dB amplification (see Figure 3.4) as well as reduced noise from both low and high frequencies. After the Butterworth filter, a diode is placed in order to ensure only positive voltages for the Arduino to read, as well as a impedance to lead away DC currents, and a capacitance to ensure no DC current is going into the Arduino. The Arduino could successfully read the pulses, although, the continuously read value was somewhat inconsistent, leading to false registered hits. To counter this obstacle an average of previous 50 values was made as a reference in order to ensure a pulse was properly registered. The built in function "analogRead(pin)"[26] however proved to be slow, taking some 100µs to read a value. The pulse sent from a Ping))) is 200µs long, and thus registering more than one sensor would not work all the times, and most of all, the time difference would be uncertain. There are ways of converting the signals to digital, but due to lack of time and no knowledge of signal conversion the idea was scrapped to the advantage of the digital modules.

3.4 Quadcopter Assembly and Regulation

This section describes the process of assembling the quadcopter and how the tracking data produced in section 3.3 is used to navigate the quadcopter.

3.4.1 Quadcopter Assembly

The first step on the path to get an automatic flying quadcopter is to assemble a basic one, without the ability to fly on its own. Then it is time to interpret the values from the tracking system described in section 3.3 to regulate the quadcopter in a way that achieves a third person view.

The vehicle used in this project is built by the components listed in Appendix A.1. The most important component is the flight controller, which is responsible for regulating the engines in such a way that the vehicle flies in a desirable direction. The flight controller that is used is the APM 2.6. It has a built in 3-axis gyro, accelerometer, barometer and a compass, which provides a lot of useful data to keep the vehicle stable and at a desired height. The major reason for choosing the APM is that it uses the open source autopilot ArduPilot. That makes it possible to modify the code and customize the way the quadcopter flies. The mounting of the flight controller should be near the centre of gravity and with a vibration damping attachment, to improve sensor readings. Further information about how to assemble a quadcopter and calibrate the sensors can be found at the Ardupilot web page [28].

To be able to manually fly the quadcopter a radio transmitter and a receiver were set up according to a guide at the Ardupilot web page [29]. A control switch on the transmitter was also configured to change from autonomous to manual mode, to make it easy for the pilot to regain control if something malfunction.

An optical flow sensor was initially used in hope of getting rid of the drifting movements of the quadcopter, when it was supposed to be in a stable position. The sensor is capturing images of the ground and compares the pixels to the previous images pixels. Then it calculates an estimate movement and tries to compensate. After a test flight it could be seen that the quadcopter was not completely stable and slowly drifted sideways. The optical flow sensor did not give any visible improvements and was therefore discarded. The negative result of improvements with the optical flow sensor can be because of the sensor's requirements of well lit environment and non fluorescent light.

The final prototype can be seen in Figure 3.11, page 36. Several different placements of the ultrasonic receivers and the ultrasonic height sensor were tested to find a place where the turbulence from the propellers and the vibrations had least effect. The best result for the height sensor was achieved at the centre with double-coated adhesive tape pads. The receivers had to be placed on extended arms to avoid the propellers and increase the resolution of the measurement of the centreline angular offset. The extended arms worked well for decreasing turbulence, but caused the quadcopter to vibrate more and impaired the flight performance. The contributing factors were probably the length of the arms and the plastic material they were made of. Another component which placement needs to be considered carefully is the compass module. It is sensitive for magnetic fields and was therefore placed at the edge of the main board to have the greatest possible distance to any electrical current.

3.4.2 Quadcopter Regulation

On the APM there are different flight modes built in, that makes the quadcopter behave in different ways. Each mode enables different types of flying, e.g Acro mode is good for acrobatics such as rolls and flips where the user is free to regulate the quadcopter as desired. While in Loiter mode the user can fly it manually, but when the sticks on the transmitter are released the quadcopter should hold its position [30]. To make the quadcopter behave in a new way a new flight mode can be constructed in the APM source code.

To regulate the quadcopter for third person view the new flight mode "Follow" was implemented. The flight mode is running a 100hz loop where the regulation is performed. This is done by PID controllers which refreshes each control signal as new sensor data is read see Figure 3.5. Until new data is received all control signals remain unchanged. The interval of new sensor data that is sent to the APM is every 60ms so the update frequency is 17Hz.

The APM uses pulse width modulation in order to send signals to the rotors. There was no need to consider how the APM controlled each engine as there were pre made methods available. A numerical value in a given interval is set for each axis that related to the velocity in that direction [31].

It was decided that the angular deviation from the quadcopter's centreline should regulate the yaw, and the difference in rotation to regulate the roll of the quadcopter (see Figure 2.3 for explanation of directions). The central argument for this is that a person can rotate quite fast, potentially causing the ultrasonic transmitters' beam angles to deviate from the quadcopters direction. Thus leaving the receivers unable to register the signals sent from the user. With angle controlling yaw this should not pose a problem. The distance to the user naturally controls the forward tilt of the quadcopter.

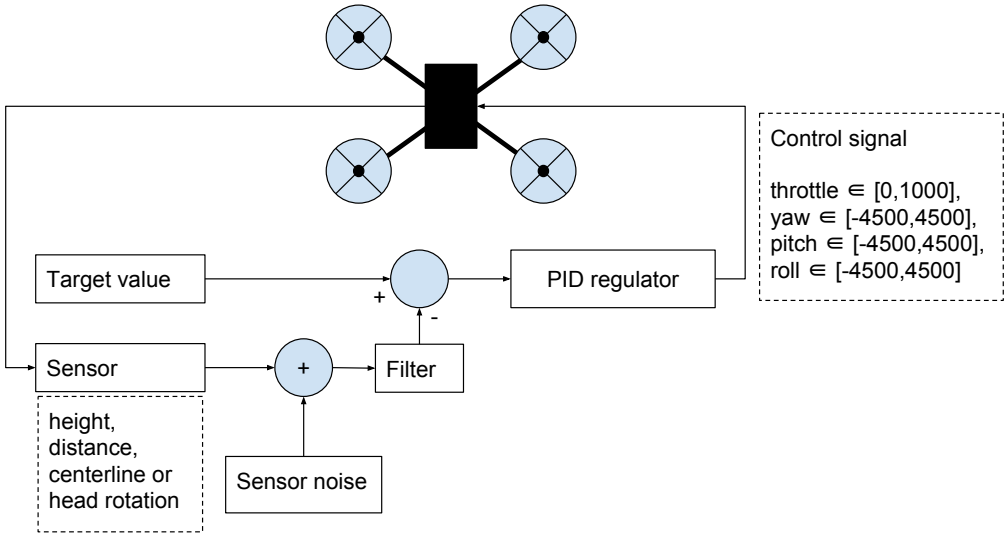

Figure 3.5: Schematic overview of quadcopter control loop. Control signals within the specified range is generated from noisy sensor data

The regulation algorithm for a user rotating is summarized in Figure 3.6. The quadcopter and user angle deviation from north represented by α and β respectively, β is the angular deviation of the user from the quadcopter centreline. As the user rotates clockwise, the quadcopter will react by rolling left induced by the fact that $\alpha < \beta$. Rolling left will generate an angular deviation θ, causing the quadcopter to rotate clockwise. A fast enough regulation will make the resulting movement a user centred rotation with a radius of 3 meters. A similar regulation will occur when the user moves sideways, but the order in which the events happen will differ. First there will be an angular deviation θ that triggers the quadcopter to rotate. The rotation causes an angle deviation from north between α and β, which initiates a rolling movement. The resulting movement will be sideways, since the distance to the user is kept constant by regulating the pitch angle.

3.4.3 PID Control

A PID controller can be either time continuous or time discrete. Continuous controllers are implemented in analogue circuits and discrete controllers in digital circuits, sometimes by software.

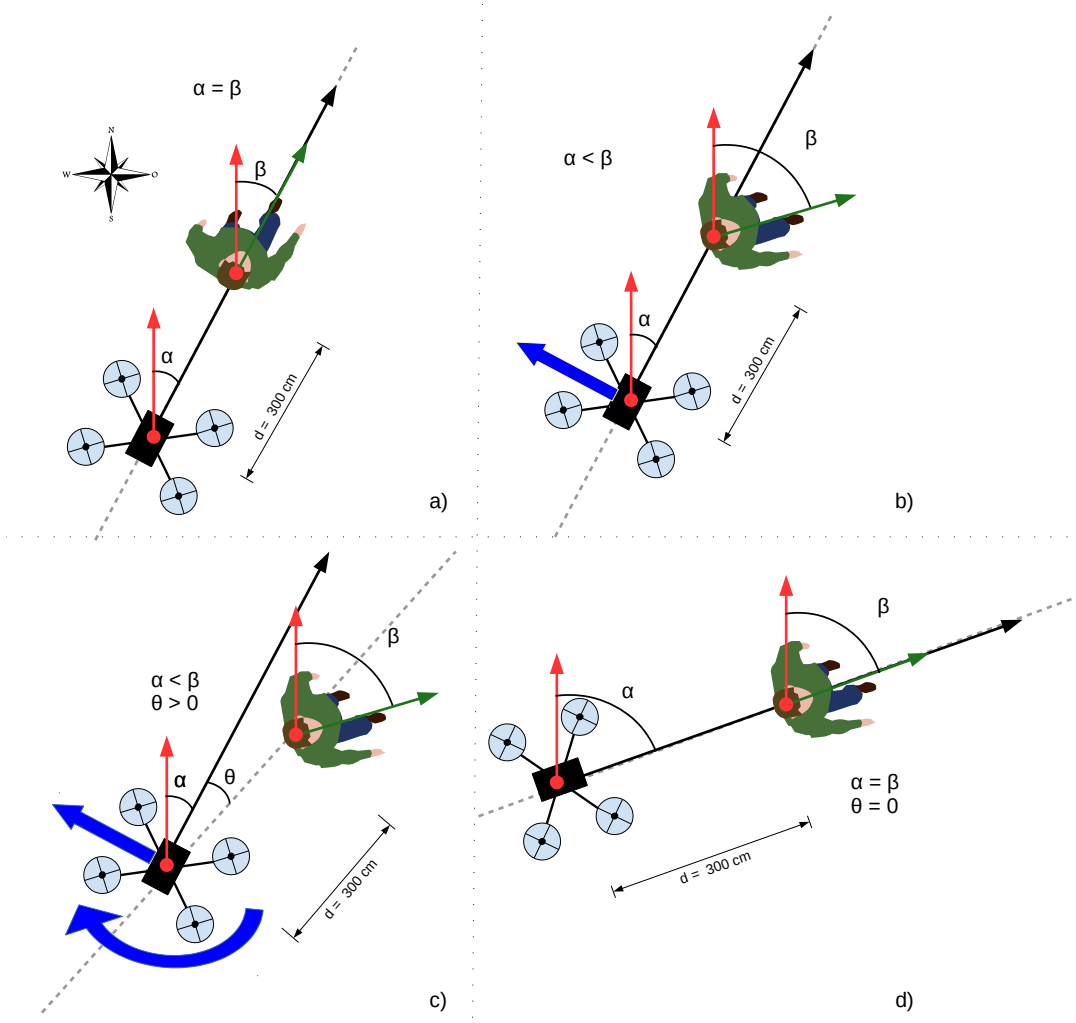

Figure 3.6: How the quadcopter reacts to a rotation of the user

At first a simple approximation of a PID controller was implemented in the APM. When new sensor values are received from the serial port the controller method updates the quadcopter control signals. This controller works the same way for all directions with different parameters, to concretize the method the height regulation will be taken as an example. The difference between measured values and target values is put in a variable e which is positive if the current height is below the target. In the following algorithm for calculating the control signal u at point k in time, Δt_k is the difference in time since the last time updating the control signal T.

$$u_k = e_k \cdot K_p + \sum_k (e_k \cdot \Delta t_k \cdot K_i) + \frac{e_k - e_{k-1}}{\Delta t_k} \cdot K_d \tag{3.7}$$

This algorithm was tried for height regulation with different K-parameters without good result. The quadcopter would, however not stable, hover for some time and then suddenly crash or fly up fast. The problem was identified as that when the height reading for some reason lagged there could go seconds between height readings. Therefore when a reading finally passed through the time passed Δt_k was very large, resulting in the sum (I-part) being very off. The PID controller was modified to not take into account the time passed under the assumption that tracking data was delivered at constant interval. This way the quadcopter did not crash during sensor data lag. A more advance variant of the discrete PID controller[32] was implemented for the height regulation. This controller is defined as follows:

$$[k] = u[k-1] + K_1 e[k] + K_2 e[k] + K_3 e[k-2] \tag{3.8}$$

where:

$$\begin{aligned} K_1 &= K_p + K_i + K_d \\ K_2 &= -K_p - 2K_d \\ K_3 &= K_d \end{aligned} \tag{3.9}$$

The implementation in C is shown in the source code provided in Appendix B.1.

The PID parameters needs to be tuned in order to function well. No simulations were done for tuning the PID parameters, rather manual tuning was made, the controller was designed to accept changes to the parameters during flight. Depending on which quadcopter movement was to be regulated, different parameters were chosen. A base value was used in order to make the height regulation (throttle) simpler, this is just a constant added to the PID controller output. The base value was chosen to be the value of the radio controller when the flight mode "Follow" was activated.

The throttle PID controller was tuned in by first increasing K_p until the quadcopter began oscillating around the target height and slowly sinking. The sinking was due to the base value being calibrated for a full battery. To compensate the error in the base value a small I-term was added. Finally a D-term was added to compensate for the oscillations.

Unlike the throttle for the other movements just a P-controller gave acceptable results. However, given more time for testing a more optimal controller with I- and D-parts could give a more responsive system. The final PID controller parameters for all movements can be seen in Table 3.1.

Table 3.1: The final PID controller parameters.

	K_p	K_i	K_d
Throttle:	0,5	0,007	7
Roll:	0.3	0	0
Pitch:	10	0	0
Yaw:	100	0	0

3.5 Information Flow and Software

In the final system which will create the third person view experience the different modules must work together. The video module can work independent of the other modules. However the tracking module and the quadcopter regulations must be tightly integrated in order to automatically position the camera correctly. Further there is a need to control and monitor the system as well as the different modules for both testing and operation purposes. This section seeks to explain the way information is passed between modules as well as the software developed to handle communication, control and monitoring of the system.

3.5.1 Information Flow

One of the main challenges is that the quadcopter regulation software ("Follow") running on the APM needs information from the tracking module to regulate the quadcopter. The different components used for the tracking are placed in different parts of the system. The orientation of the user and the quadcopter come from compasses on the DK2 and the APM respectively. The distance and the angular offset between the user and the quadcopter is given by ultrasonic sensors connected to the Pi as presented in section 3.3.2.

The regulation software Follow regulates quadcopter movements by adjusting control parameters for yaw, pitch, roll and throttle. A discussion was where the conversion from sensor data to control signals should take place. The main intention that determined the choice was minimizing complexity in the information flow. The orientation of the quadcopter is needed for regulation, which is directly available in the APM API. This makes it logical to pass all sensor data directly to the APM. The alternative would have been to send information back and forth between the Pi and the APM for calculations on the Pi, but that would require a two way communication link.

The tracking data originates from both the laptop with the DK2 as well as from the tracking module on the Pi. Since a Wi-Fi link between laptop and the Pi was already established for the video module, one of the available options was to use it for transmitting tracking data to the Pi. The Pi now have all needed data to send to the APM on a single link.

3.5.2 In-Air Microcomputer Software

The Pi on the quadcopter handles both the tracking module as well as the capture and transmission of the video. There is a need to be able to receive the sensor data from the DK2, sending all the sensor data to the APM as well as offer control over the modules. To offer this functionality a software has been developed called piApp.

Its main component is a TCP/IP server module receiving connections from the laptop over Wi-Fi. The piApp server offers an API for clients to control both video capturing and tracking as well as regulation settings. This is also how tracking data from the DK2 is received.

The interface uses JSON formatted strings, sent from the clients connected to the server. The piApp decodes the JSON strings to different commands. The available commands is listed in Appendix B.2 and includes settings as well as tracking data updates.

The piApp has been developed in the programming language Python 3. One of the main reasons is that the implementation of a threaded asynchronous TCP server is fairly easy in Python. The reason a threaded server is needed is because of the time critical operations piApp executes. Among these are the use of the tracking module to get the sensor data and the transmission of the sensor data to the APM. Further the tracking module has also been developed in Python to make it easy to integrate with the server module. Python also offers an easy native way to use serial communication which is used to communicate with the APM.

The APM has support for both I2C (Inter-Integrated Circuit) and serial communication. The I2C port at the APM was used for the compass module and therefore serial communication with the unused GPS port was chosen.

Receiving data is done in a method running by 50 Hz. Messages are not overflowing the incoming 256 byte buffer as reading is done more often than new tracking data is sent. Messages only need to be sent when new tracking data is available, which is at a 17 Hz frequency.

A problem was that the APM and Pi had different working voltages, 5 V and 3.3 V respectively. To overcome this at first a regular Arduino was connected between the Pi and APM only passing messages, as this supported both voltages. However this for unknown reasons was not a stable solution and messages was often distorted causing the APM to halt. Instead a Logic Level Converter was bought that enabled the Pi and APM to communicate directly by serial communication.

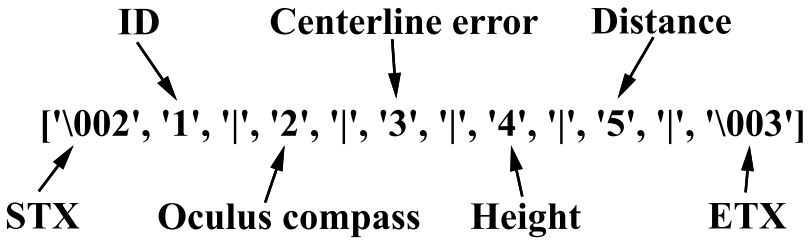

Figure 3.7: An example of a message to the ardupilot containing sensor data.

A communication protocol was developed and a typical message to the APM can be seen in Figure 3.7. The message is encoded in ASCII and the start and end of the message are identified by the standard control characters STX and ETX. The first number works as an identifier, which tells the APM how many and what type of values to expect. Each value is separated by a divider character ('|'). Apart from updates of tracking data, which is the most common message sent, there are messages for updating target height and PID-controller parameters as well as enabling and disabling parts of the automatic control. One option is for example to only regulate height automatically and control the other axes with an RC controller.

3.5.3 Software for System Monitoring and Control

The Third Person View Control Centre (TCC) is a developed software component that offers a graphical user interface to observe and control most parts of the system. It also handles extracting the DK2 sensor data. TCC connects to the piApp server on the Pi and offers a user interface to the functionality that piApp offers. It is intended to mainly run on the user worn laptop which the DK2 is connected to. Under normal operations it must send sensor data to the piApp and starts up GStreamer for the video link. Extra instances of TCC can be run on other machines in order to observe the system and get access to the system controls on separate machines.

The DK2 sensor data is provided by the DK2 SDK. In order to easily access the sensor data the TCC as the SDK was implemented in C++. Further, since the intention of the TCC software was to offer a user interface the QT platform was used. The QT platform is a C++ framework offering extensive functionality for creating user interfaces [33]. QT also offers several modules, among them a module for network communication. This made it easy to connect as a client to the TCP/IP server that piApp runs.

The user interface of TCC is extensive and offers a way to both observe and control the system. The main features of the user interface are to make it easy to manage the connection to the piApp server and provide piApp with user orientation data from the DK2. It can also receive sensor data from the piApp and graph this. There is further a possibility to change the quadcopter PID controller parameters and other regulation settings. All this data is packed in a JSON string and sent to the piApp. The user interface is further explained in Figure 3.8.

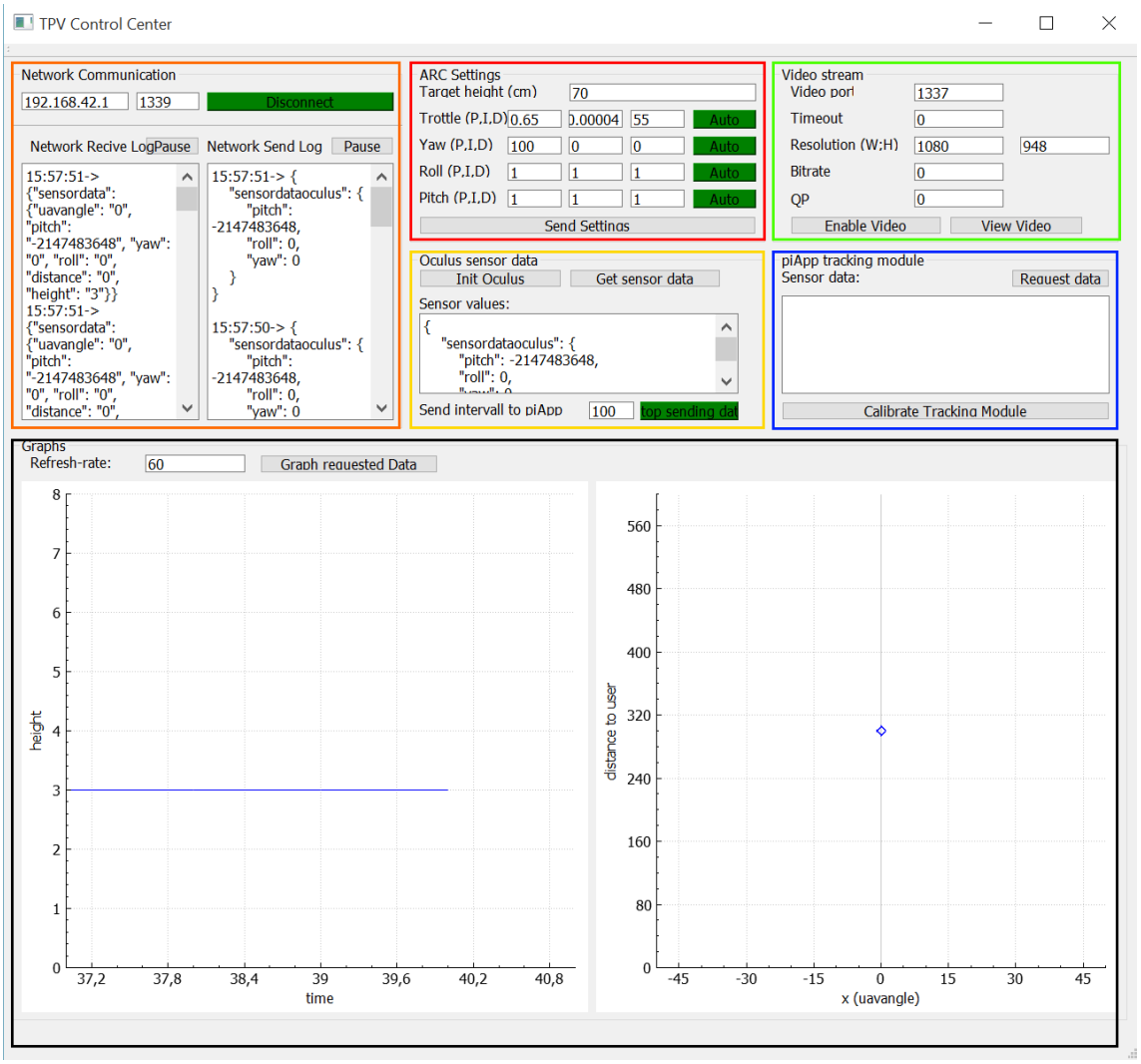

Figure 3.8: The GUI of the TCC software.

The orange box indicates the controls for connecting to the server running on the Pi. It also offers a network communication log. The red box is the GUI for sending settings to the software on the APM, mainly PID parameters. It also allows for choosing what directions should be manually controlled. The yellow box offers an interface towards the DK2s sensor data and the frequency it should be sent to the piApp can be adjusted here. The green box is where the settings for the video stream can be adjusted. The blue box displays the sensor data available on the Pi for debug, testing and observation purposes. The **black** box further graphs sensor data on the Pi, the left graph shows the height readings over time while the right graph shows a combination of distance and user centreline offset.

3.6 System Overview

This section will give an overview of the final prototype. An overview of the prototype with the different modules and their communication links are presented in Figure 3.9.

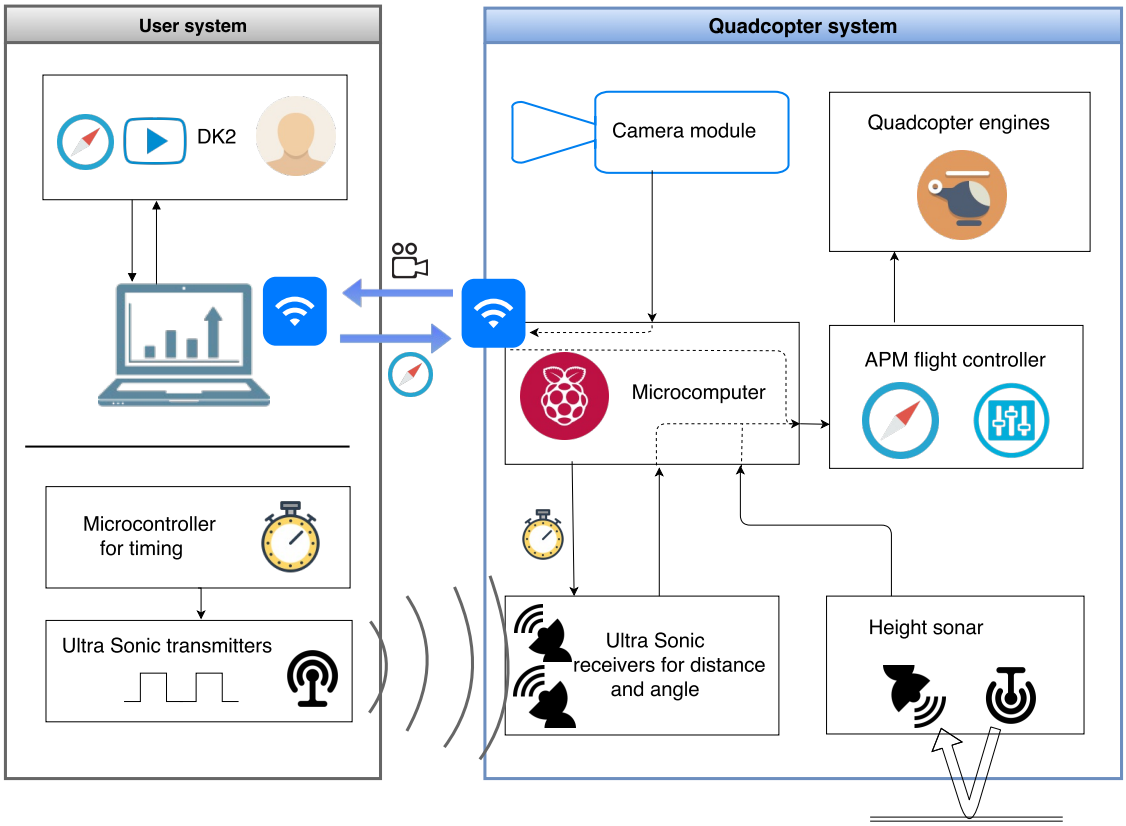

Figure 3.9: Overview of the components in the system and their communication links. The large boxes shows division between system components worn by the user and components mounted on the quadcopter frame.

The user will carry the head mounted display Oculus Rift Development Kit 2 connected to a laptop as well as a ping transmitter. The DK2's inbuilt compass is used for collecting the user's current head rotation. The DK2 is further used for presenting the video stream. The laptop handles radio communication with the quadcopter; sending compass data from the DK2 and receiving the video stream. This communication is done over 802.11 Wi-Fi. From the laptop there is also possible to change settings for the automatic control system on the quadcopter. The ultrasonic transmitter is made up from a battery driven Arduino connected to an ultrasonic ping module. The Arduino's internal real time clock is used to make the module send ultrasonic pulses by a constant 60 ms interval. This equipment is to be worn at the back head of the user so that the pulses reaches the receivers on the quadcopter as per Figure 3.10.

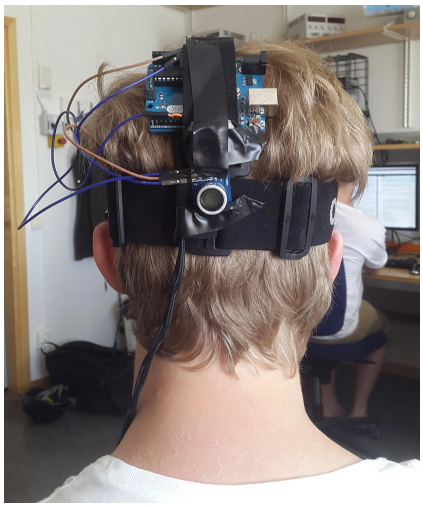

Figure 3.10: The ultrasonic pingtransmitter worn at the back of the user's head

The quadcopter has many different components, and they are mounted according to Figure 3.11. The microcomputer Raspberry Pi acts a central coordinating unit. The Pi communicates with the user worn laptop by broadcasting a Wi-Fi network. The Pi also receives height measurement data from the ultrasonic distance sensor directed towards the ground. At a 60 ms interval it activates the two other ultrasonic receivers as described in Figure 3.1 on page 17, which will sense the pulse transmitted from the user. The distance to the receiver from the transmitter is given by a simple calculation from the travel time of the pulse. By comparing the measured distance from each receiver one can determine the users centreline offset. For measuring distance the Pi software must know when the pulse was transmitted. At an initial stage the internal clock in the Pi is synchronized with the clock on the user side Arduino microcontroller.

All track data is sent by the 60 ms interval from the Raspberry Pi to the APM flight controller over a serial data connection. The order of tracking data and/or setting data is according to a predefined protocol which minimizes overhead. The APM runs a constructed control system which first filters the incoming data to remove potentially incorrect tracking values. The reported height is adjusted for the quadcopter's inclination as shown in Figure 3.2.

The filtered sensor data is used by PID regulators, which generate control signals according to Figure 3.5. The control signals adjust the movement of the quadcopter to track the user as in Figure 3.6. A compass offset triggers a sideways motion (roll), a height offset triggers a throttle adjustment, an angle offset triggers a rotating movement and a user distance offset triggers forwards or backwards movement (pitch).

The real time view system is what creates the third person immersion when the camera is positioned. For capturing the video a Raspberry Pi Camera module is used. The camera module is connected to the Pi, where the video is captured and converted to a raw h264 stream at requested dimensions and quality using hardware encoding at the GPU. This functionality is made possible by the Pi's official Raspivid software.

The h264 stream is encapsulated in RTP packets and transported to the user worn laptop by UDP/IP. This is done with GStreamer using a custom pipeline of existing modules for encapsulation etc. The video stream is transmitted over the same Wi-Fi network as the one for receiving tracking data. At the laptop side the stream is unpacked also by a GStreamer pipeline and duplicated so that a side-by-side image is finally presented on the DK2 acting as an external monitor.

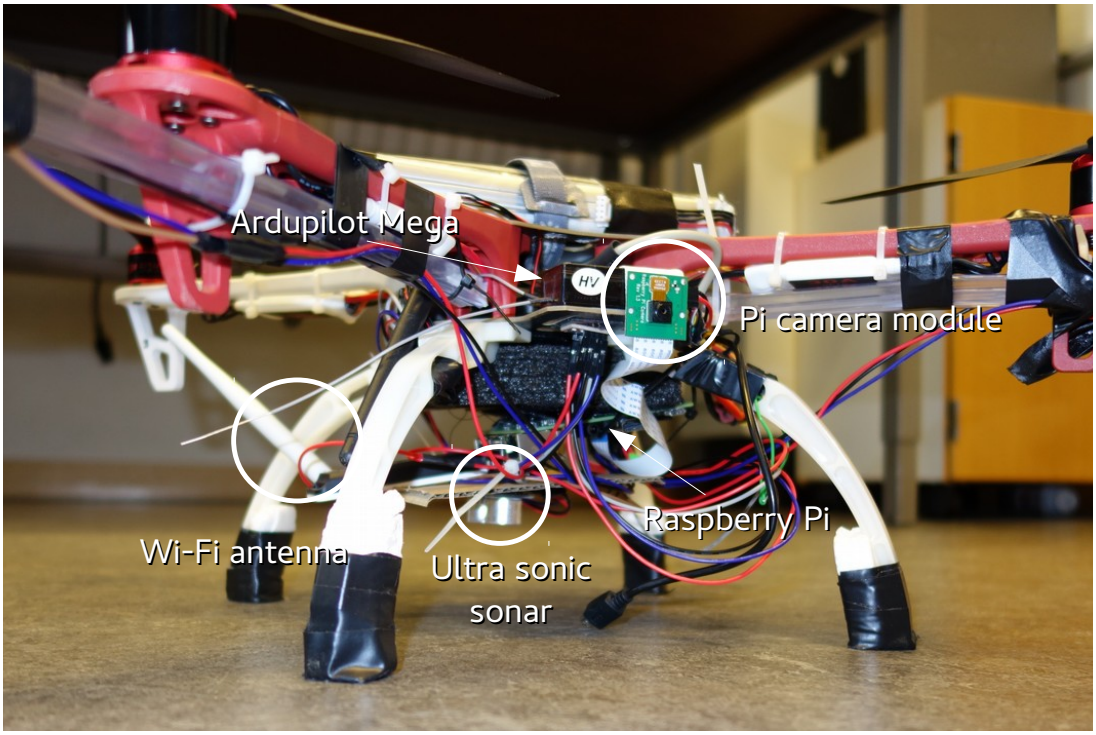

Figure 3.11: The assembled quadcopter with some components marked. The ultrasonic receivers extend outside the image

4

System Tests and Performance

In this chapter test results are presented for the video link and the different sensor systems as well as the regulation software. The tests have been performed on individual components separately as well as together in flight through different flight tests.

4.1 Video

The video latency was tested by filming a digital clock with the video capture device and then taking still photos with a high speed camera comparing the times at the clock and displayed video. An example of such a photo is in Figure 4.1. Known error sources are the refresh rate of the monitor used which was 60HZ. Further the timer used might not be exact. In Figure 4.2 latency was compared for UV4L and Raspivid with GStreamer. The Wi-Fi link was not throttled. Other settings is 42 frames per second, h264 used 30 as quantum parameter for compression and MJPEG used compression level 5. The results are the average latency derived from several stills.

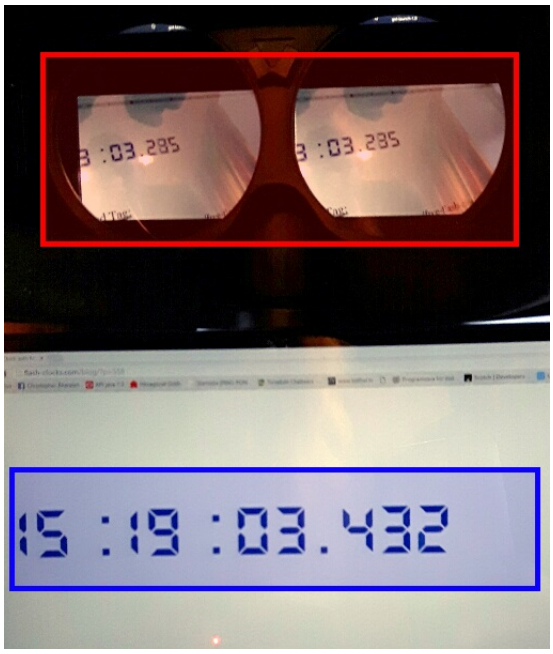

Figure 4.1: Example of photo used for testing latency. It is a single photo taken of the digital timer and the rendering device (DK2) at the same time.

On the picture the delay is shown. The blue box contains the reference time, a timer running on a laptop. The red box contains the reference time and latency difference as rendered on the DK2.

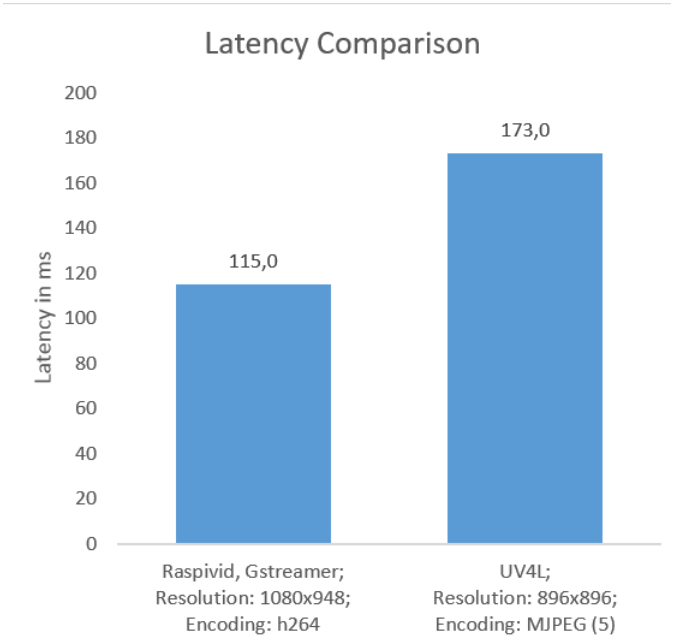

Figure 4.2: Video latency comparison between Raspivid+GStreamer and UV4L over Wi-Fi with UDP. It shows the average latency over 20 stills.

4.2 Ultra Sonic Sensor Performance

The ultrasonic sensors are responsible for measuring height, user distance and user centreline offset. They where tested when the quadcopter was standing still with the transmitter at a constant distance. This gives information of the long term stability of the system and variances in measurement results. The results for eight minutes of measuring is presented in Figure 4.3.

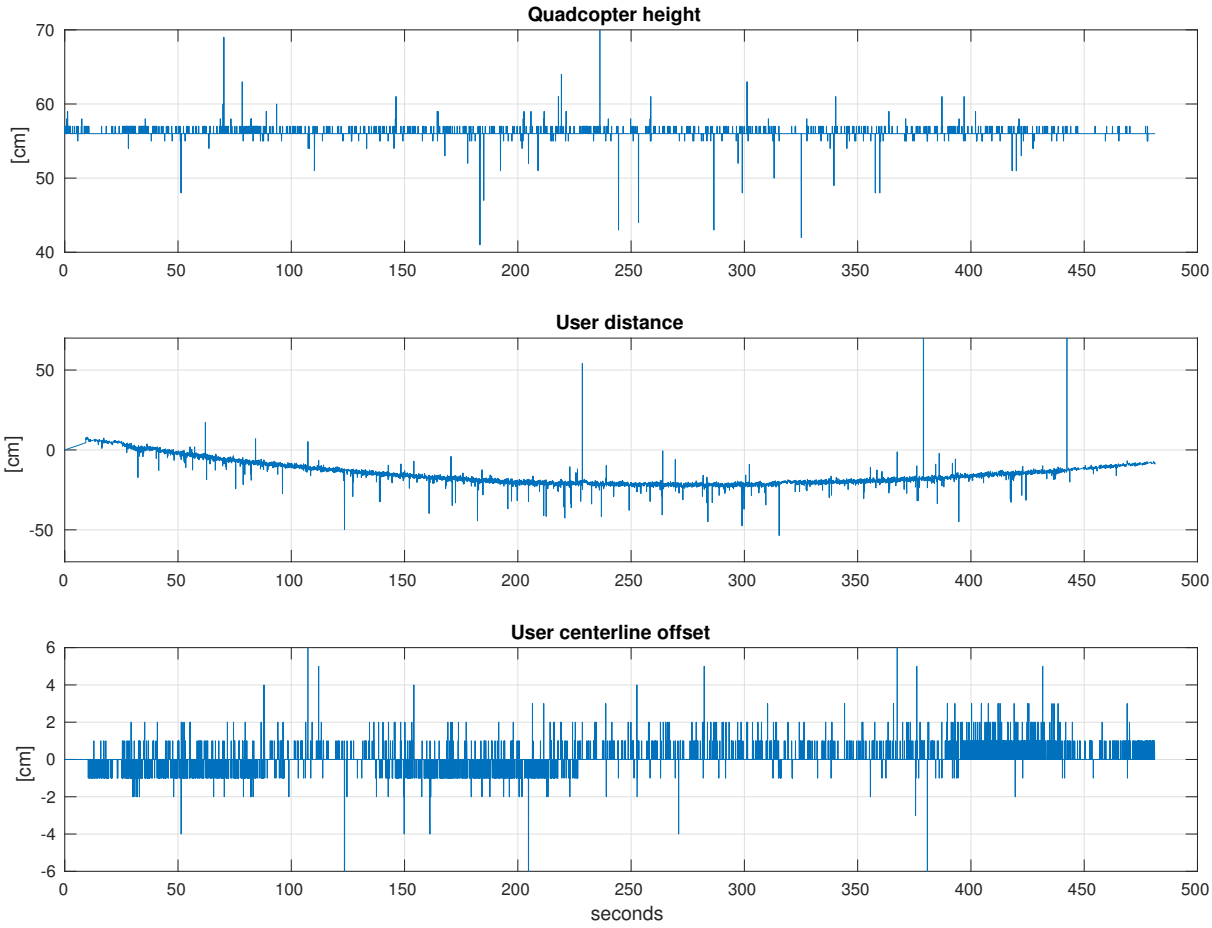

Figure 4.3: Sensor data during 8 minute ground test. Graphs sensor data from the Ultra Sonic Sensors.

There are several interesting things to note here. Regarding the height the reading is not drifting, and the standard deviation is about 5 cm. There are several spikes occurring regularly. However, the height sensor data is filtered before regulation according to the method in section 3.3.5.

The user distance is measured when the drifting of the internal clocks was taken in consideration as discussed in section 3.3.3. However, it is evident that the clock drifting is not constant, it is still drifting, but it is not linear. The standard deviation is 8.6 cm.

The centreline offset is rather constant. The centreline offset standard deviation 0.76 cm. Why it is not entirely stable is uncertain. It is also worth to note the relatively low resolution of 1cm.

Further tests were also performed for the maximum detectable user angle Θ as in Figure 3.1. Since the ultrasonic sensors have a narrow sensing angle the quadcopter will not be able to track the user if it deviates from its steady state. In order to test this the transmitter was placed on a known distance and moved in a line parallel to the quadcopter. When the receivers stopped receiving a signal the distance was noted and the angle measured. The result was that the receivers worked at an angle of incidence of up to 23° with regards to the transmitters.

4.3 Flight Tests

Flight tests were conducted to see how the quadcopter behaved in air. The quadcopter needs to regulate the different directions throttle (height), pitch (distance), yaw and roll. These were tested first separately where the PID-parameters for each direction were set to what was found as optimal. Then they were tested together in different configurations. Lastly the system was tested while all directions were regulated automatically.

4.3.1 Test Round 1: Altitude Hold Performance

The height was tested by visually observing if the quadcopter were able to hover stable on its own. Further testing is required for getting optimal PID-parameters. No tests were made where the target height was altered in order to see how it behaves to changes. This would be required to optimise the PID-parameters. The quadcopter is successfully capable of holding its height with the PID-parameters found in Table 3.1. The current algorithm does not allow significant changes to the target height, due to the d-parameter reacting aggressively to a sudden change in target height.

4.3.2 Test Round 2: Distance and Height Hold Performance

The next test round involved testing the distance regulation. While testing the distance the quadcopter regulated both the distance (pitch) and the height on its own while the pilot adjusted the other directions. The quadcopter began at the desired distance from the user, the user then performed a movement towards the quadcopter. The reaction of the quadcopter was observed. Interesting observations was how fast it reacted and how long time it took to reach a steady state again. Test data from one of the tests is presented in fig 4.4 where two movements, or thrusts were made. The quadcopter had a hard time stabilising after a movement, both slow and fast movement. Regarding the height it seems to be able to hold the height despite the rapid movements. Worth to note is that the figure presents the height values before they are adjusted for the orientation of the quadcopter.

It was not trivial to regulate the PID-parameters to minimise the time for the quadcopter to stabilise after movement. The reaction time from a movement of the user was however very good. The final PID-parameters used for the pitch is presented in the PID-controller parameter Table 3.1.

4.3.3 Test Round 3: Yaw and Roll

The yaw is adjusted with the data from the centreline offset. The yaw regulation was tested by simply letting the user walk around the quadcopter and see how well the quadcopter rotated aiming at the user. The roll was similarly tested by letting the user rotate the DK2 and then observing how well the quadcopter follows by regulating its roll. The roll and yaw seemed to regulate well, however further testing is needed to see how fast it reacts and how long time it takes to stabilize after movement with regards to yaw and roll. The final PID-parameters used for the roll and yaw is presented in the PID-controller parameter table 3.1.

4.3.4 Test Round 4: Full Auto

In the last test round the quadcopter regulated all four directions: throttle, yaw, roll and pitch. The results vary a lot and while the quadcopter handled a steady state well, different types of fast movements from the user could lead to irregular behaviour. A lot of further testing is needed to see if the different PID-parameters can be optimized for better performance. One of the main theories for irregular behaviour is that the quadcopter cannot react fast enough to user rotation, meaning that it no longer gets sensor data from the centreline offset. This can be solved by increasing the angle presented under Section 4.2.

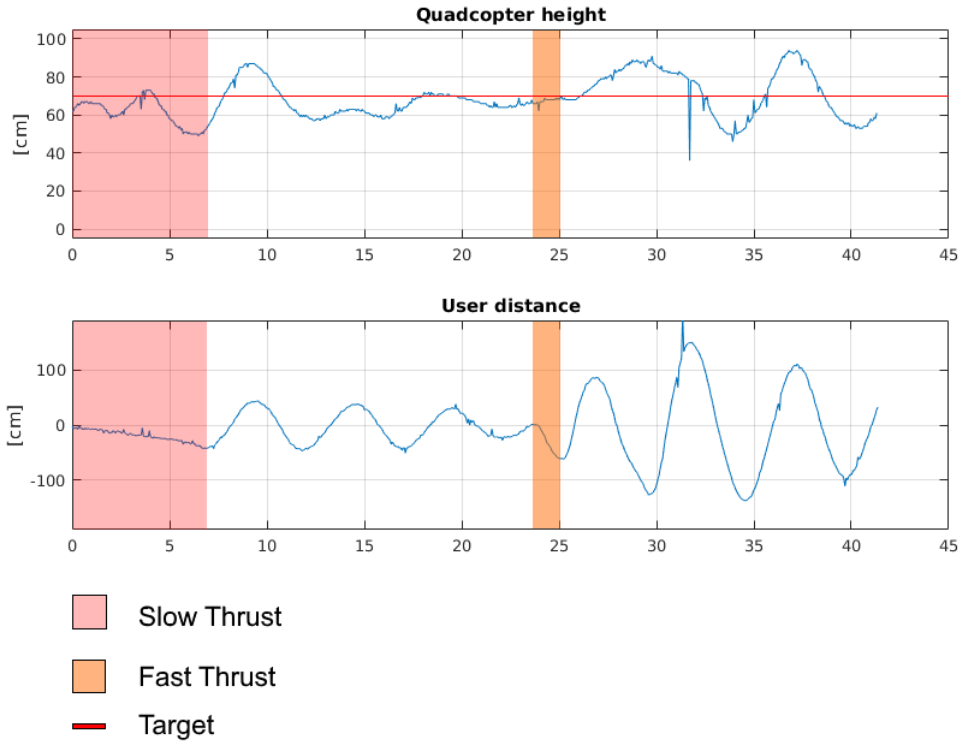

Figure 4.4: Sensor data for a test with throttle and pitch automatically regulated.

The upper graph indicates the measured height (before adjustment for quadcopter orientation and without filtering) where 70cm is the target height. The bottom graph shows the distance from the user, where 0 is the target distance which is the deviation from 3 meters. The highlighted areas of the graphs is where a thrust where made towards the quadcopter by the user.

Sensor data from one of the tests is shown in Figure 4.5. Unfortunately the logging does not support sensor data from the DK2 and the compass in the flight computer. It shows an example of about 20 seconds automatic flight. During this particular test the user first took some step backwards and then rotated 90 degrees slowly counter-clockwise and then 90 degrees fast clockwise. When rotating clockwise the quadcopter lost tracking data from the centreline offset and the pilot landed the quadcopter with manual control. This is shown in the graph where the distance and centreline offset show 0. The oscillation present from 30 seconds forward started after the user took two steps backwards. Trying to correct for the about 2m distance movement was enough to make it hard for the quadcopter to stabilize.

It is clear that further testing and improvement is necessary before the quadcopter can regulate well for user movement. As of this time it can only handle very slow movement with regards to user rotation, and fast difference in distance causes it to be unstable for an unfavourable amount of time.

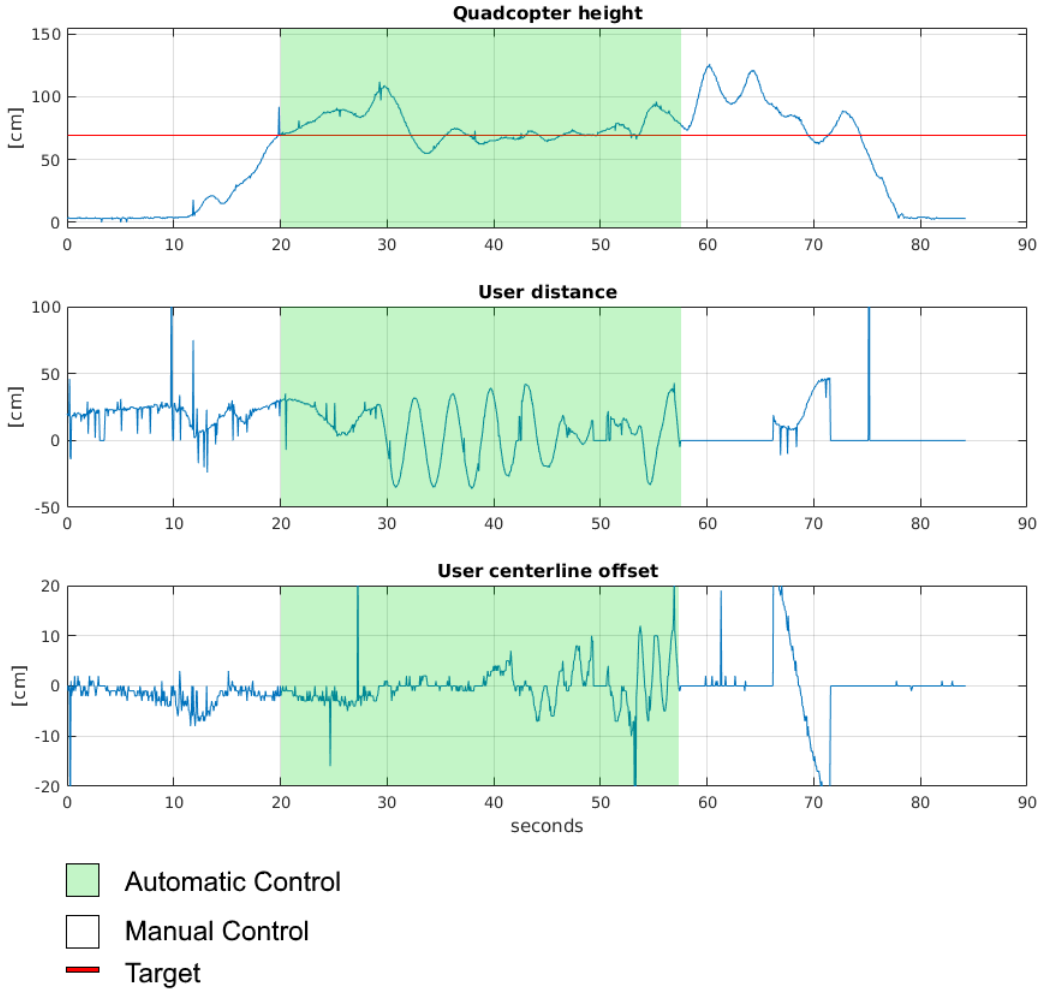

Figure 4.5: Sensor data from a fully automatic test.

The figure shows the sensor data for height, distance and centreline offset from the full auto test. The green highlight indicates the time when the quadcopter was regulated automatically. For take-off the pilot had control, and when the quadcopter lost sensor data for distance and user centreline offset due to fast movements from the user, the pilot took control again towards the end of the testing. The red line indicates the target height. The target distance and target centreline offset is 0 and is the deviation from where the quadcopter should be.

4.4 Analogue Circuit

The analogue ultrasonic amplifier circuit developed as in section 3.3, which can be seen in Figure 3.3 on page 24, was built on a breadboard to see how well it did its job. The theoretical amplification of the first stage amplification and the active 4th order Butterworth bandpass filter for 40kHz is supposed to be 66.7dB. But it was measured to 52.7dB. After the diode and DC isolating capacitance the amplification should be the same, but it was measured to a lower amplification, 49dB. See Table 4.1 for conversion.

Table 4.1: The theoretical and measured values for a 40kHz signal given by the circuit seen in Figure 3.3

Type	Amplitude [dB]	Gain
Theoretical value	66.7	2163
Measured value before diode	52.7	432
Measured value for Arduino input	49	282

5

Discussion

In this section the results, progressions and development will be discussed. Suggestions on further improvements and alternative methods will be made to make it easier for future projects as well as to ascertain what went right and wrong for self-evaluation.

5.1 Regulation

The regulation was tested sparsely, but it does work. The quadcopter is responsive and the regulation software worked well. However, further optimization of the PID parameters is necessary, and that is not difficult considering how easy it is to upload new PID parameters thanks to the software's developed. Some of the reasons the regulation is not working as good as intended, apart from the lack of testing and optimization, is because of the limitations in the ultrasonic measurement.

5.2 Ultrasonic Tracking

The use of synchronized ultrasonic sensors to measure relative distance and angular offset worked rather well. The sensor values were adequate enough to use to regulate the quadcopter, but the tracking system had a few flaws. One problem that emerged was the short term stability of the processors clocks. This caused the quadcopter to drift the distance to the user. The attempt of using linear compensation gave a positive effect on the distance measurement and made the tracking work well for relatively short flight times.

The drawback is that the compensating clock constant has to be set at each flight occasion as the amount of clock drifting varies. Also calibrating the clock constant is not enough to fully compensate for the clock drifting, as it is not even short time stable. As can be seen in Figure 4.3 of the Result section the distance reading is not stable even with a perfectly calibrated clock constant. The sensor error varies by up to half a meter during the eight minute sensor test. For a more accurate reading external high precision clocks might be used both at the Arduino and Raspberry Pi, although there was not time to test this. At a minimum automatic calibration of the constant should be implemented.

Another unforeseen issue was the turbulence from the rotors affect on the ultrasonic sensors. The extension arms for the sensors on the quadcopter greatly reduced the interference from the rotors. Unfortunately the extension arms oscillated during flight, leading to decreased flight performance. It further gave the quadcopter unbalanced mass, negatively affecting regulation of the pitch movement. This could potentially be solved with different material for the arms and counterweights or a redesign of the ultrasonic placement entirely.

The ultrasonic sensor measurement worked well for slow user movements where the quadcopter always was kept in the steady. During rapid movements the quadcopter moved out of the sideways range of the ultrasonic transmitters. This is partly because of the limitations in the receivers and transmitters beam angle. In order to achieve greater angle an increase in signal amplification might be needed for both transmitter and receivers [34]. This can be accomplished by using stronger transmitters as well as more sensitive receivers including amplifiers, possibly by them by yourself. The attempts to create amplifier circuits to be used with wider ultrasonic elements failed but should be possible.

Another option for improving the angle is to use an array of several synchronized transmitters and receivers. Individually only listening/transmitting in a narrow direction but combined providing a 360deg angle. An improved tracking angle would decrease the chance of the quadcopter moving out of range of the sensors. To finetune the PID parameters for the yaw and roll to keep the quadcopter closer to steady state would help the quadcopter to not moving out of range of the sensors but would probably not be enough.

Regarding the height measurement it is only accurate for low heights. Even at as low as 2 meters the measurement quality was unsatisfactory. One of the factors could be that the sound have to travel further at higher altitudes making it more prone to disturbances such as turbulence. However, for a third person view experience heights over 2 meter might not be relevant.

Apart from the issues presented there was also some irregularities detected in the form of "hickups" where the tracking module would freeze for some time, or skip some measurement loops. The use of a Raspberry Pi for ultrasonic measurement might be one of the sources for these errors. A microcomputer with an multitasking operating system is running several different processes, thus leaving the measurements vulnerable to errors caused by an undedicated processor. Further it is not made for real time critical operations. Thus a dedicated microcontroller might serve better, such as an Arduino.

5.3 Real Time View

The initial goal was to implement, and then improve upon the solution that J. Allander et al. [1] used. It turned out that it was difficult to replicate their solution. One of the main reasons is that their solution is largely undocumented. Instead we experimented with alternatives using the same hardware set-up. Unfortunately the results were below our initial goal with regards to latency with 115ms as presented by Figure 4.2. As the problem with low latency had already been solved in an earlier project limited effort was made to improve the video streaming and more time was spent on other project parts.

We also carried on using the DK2 as an external display displaying the video but without doing the rendering through the DK2 SDK as per J. Allander et al [1]. This means that there are image distortions not handled so that the experience is arguably impacted. However, it remains uncertain if rendering through the DK2 SDK can be done with low enough latency. We believe that it would be better to work with designing modules for the GStreamer pipeline that adds filters to the video stream to compensate the DK2 induced distortions.

Regarding the video quality we reached good results. The resolution is at DK2 maximum and the compression is good enough for the video to be sent over the Pi's relatively low bandwidth Wi-Fi link. It is our opinion that the video quality is good enough for high immersions with the DK2. The DK2 only supports a 1080p resolution which is in itself a limiting factor. J. Allander et al [1]. uses MJPEG with a rather high compression level leading to worse image quality than we achieved. It is possible to reach even higher video quality with h264 but that would require improving the bandwidth of the Wi-Fi link. One option might be to use another microcomputer, maybe one with inbuilt Wi-Fi.

For the DK2 a laptop was used. The laptop was impractical for the user to carry around. Attempts where made to use the DK2 with other hardware such as a Pi but without good results. DK2 has no official support for other hardware than computers running Windows with a dedicated graphics card. It is worth to consider if the DK2 is simply too unsuited for mobile operation. A possibility might be to use regular smart phones with VR head mounts for a more mobile experience and for some phones even getting higher resolution.

Over all the real time view performs okay, but below our expectations and requirements. There are various reasons for the results. As mentioned the DK2 is not suited for mobile operations. The alterations of the video stream that has to be done in order to get a good experience on the DK2 is also a problem. It is also clear that it is very difficult to reach low latency results with a completely digital system.

The camera used was the Pi camera module which has limitations in the amount of frames per second it can record video. Upgrading to the newer Raspberry Pi HD camera module which can record at 60 frames per second at 720p would be a nice increase in quality and would heighten the user experience if the bandwidth problem is solved.

5.3.1 Analogue Circuits

The tested analogue circuit was working, the Arduino could register the values. But due to the time cost of reading an analogue value the idea was scrapped. A viable option is to implement a solution in order to read it as a digital value. A digital read is much faster. The circuit also produced a much lower amplification than expected. It might partly be because of the test being done on a breadboard. This is to be considered when creating analogue circuits. The used resistances, capacitors etc. was not of high performance quality, which probably would improve the real amplification. Our chosen filter, the Butterworth band pass, might not be the most efficient. There are a lot of different characteristics of different filters and chosen cut off frequencies to be considered. Overall, the circuitry requires a lot of knowledge and insight into the world of electronics. Choosing components might not yield a great result unless one is informed of what is important.

5.4 Ethical and Economical Aspects

The technology presented in this thesis is under rapid development. As Brynjolfsson and McAfee presents in their book "The second machine age" [35], the technical advancements move very fast, and with that the technology becomes cheaper. The first version of the Raspberry Pi was released 2012 [36] and DJI, one of the leading companies in video capable quadcopters, was founded in 2006 [37]. 10 years ago (2006) this project might have been doable, although much more expensive and most certainly a lot harder. The progress is ongoing and it is easy to believe that all the products used will be more common in the future.

The possibilities of this implementation are vast. As mentioned in section 1 there are several benign possibilities for using this system, or parts of it. It is probable that it can be profiteered on, there are already today complete systems for live views from a quadcopters perspective. The quadcopter together with the view could be used for recreational purposes, or decrease work accidents in dangerous environments, such as burning houses, mines and nuclear reactors.

There are also a risk of someone using the technology presented here for maliciously intended purposes. If the video stream would be hijacked, the integrity of the user is compromised. It could also become monitored by a government in order to accumulate data. A worse scenario would be if the quadcopters system would be hacked. In our implementation it's possible to load up new parameters for the regulation, and someone could change them in order to hurt the user or someone around. The drone creates a lot of sound, if used in public, it could be perceived as noisy, as well as threat. The drone could easily hurt persons and property by a crash.

5.5 Future Recommendations

We recommend future works to change video capture and presentation system. There are a few complete system on the market like the Fatshark, which are wireless and promises low latency.

As for the height measurement it might work better using the method used for the centreline offset. Placing two receivers distanced vertically from each other, the quadcopter can regulate to be on the centreline between the two. Using a barometer together with accelerometers might also work, it would not get the error values caused by wind turbulence and sound from the rotors. Another solution would be laser which has very good resolution [17] and would not be disturbed by turbulence, but it is rather expensive.

The centreline offset measurement had an interval between -23 to 23 degrees. This limited how fast the user could move since the quadcopter have to regulate so the user will stay in this interval. A solution for this is to use ultrasonic sensors that have a wider beam angle. Also sensors with longer distance measure would decrease measurement failures. An improvement would be to use a digital receiver that can register a hit any time. This enables a higher resolution for the centreline offset, but leaves no solution for distance.

Some type of filter should be used on the regulation, in order to decrease measurement errors impact, to improve the overall stability and response time.

Further development to detect and avoid unforeseen objects or structures is a topic not touched in this report. Further advancement could manage people walking in between the user or follow the user alongside a wall, even if it is "supposed" to hit the wall. This would be the next step to get a usable product.

5.6 Wrap-up

The subproblems presented in chapter 1.3 Introduction were all solved to some degree. The Real Time View presented video with a latency of 115 ms, performing slightly below the initial goal of at most 100ms. The method of ultrasonic tracking together with the regulation of the quadcopter looks very promising with fully automatic tracking possible for a slowly moving user. As the result from section 4.3.4 shows, when the user makes even modestly fast movements the tracking system can lose track of the user completely. Further improvements to make it more robust is needed; with ideas for so have been discussed. That being said, the final system performs rather well, showing potential of delivering an immersive TPV experience.

Bibliography

[1] J. Allander, A. Garpetun, A. Hahlin, J. A. Hultén, J. Karlsson, and A. Käll, "Tredjepersonsvy i verklig miljö: videoströmning från autonom luftfarkost", Chalmers, Tech. Rep., 2015.

[2] A. Aronsson, A. Gashi, B. Karlsson, T. Olausson, M. Patricks, and A. Wallström, "Autonom kortdistansnavigation i öppen miljö: utveckling av kameraföljningssystem med drönare och oculus rift för tredjepersonsvy i realtid", Chalmers, Tech. Rep., 2015. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/219011/219011.pdf.

[3] A. Lindgren, D. Göransson, E. Snellman, J. Hagstedt, P. Suorra, D. Larsson, and J. Andersson, "Third-person immersion: vision-based autonomous target tracking for unmanned aerial vehicles", Chalmers, Tech. Rep., 2014. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/203230/203230.pdf.

[4] J. Nielsen, *Response times: the 3 important limits*, 1993. [Online]. Available: https://www.nngroup.com/articles/response-times-3-important-limits/.

[5] Oculus, *Introduction to best practices*, 2016. [Online]. Available: https://developer.oculus.com/documentation/intro-vr/latest/concepts/bp%7B%5C_%7Dintro/.

[6] M. Abrash, "Abrash dev days 2014", *Valve*, 2014. DOI: http://media.steampowered.com/apps/abrashblog/Abrash{\%}20Dev{\%}20Days{\%}202014.pdf. [Online]. Available: http://media.steampowered.com/apps/abrashblog/Abrash%20Dev%20Days%202014.pdf.

[7] Stardot Technologies, *Stardot bandwidth and storage calculator*, 2016. [Online]. Available: http://www.stardot.com/bandwidth-and-storage-calculator.

[8] D. Pohl, G. S. Johnson, and T. Bolkart, "Improved pre-warping for wide angle, head mounted displays", in *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, ACM New York, NY, USA ©2013, 2013, pp. 259–262. [Online]. Available: http://dl.acm.org/citation.cfm?id=2503752.

[9] Ardupilot Dev Team, *What is a multicopter and how does it work? — copter documentation*, 2016. [Online]. Available: http://ardupilot.org/copter/docs/what-is-a-multicopter-and-how-does-it-work.html.

[10] B. Lennartson, *Reglerteknikens grunder*, 4th ed. Lund: Studentlitteratur, 2002, ISBN: 9789144024165.

[11] *Raspberry pi camera module specifications*. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/camera.md.

[12] Raspberry Pi Foundation, *Camera module*, 2016. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/camera/README.md.

[13] D. Jones, *Rapid capture and streaming*, 2014. [Online]. Available: https://picamera.readthedocs.io/en/release-1.10/recipes2.html%7B%5C#%7Drapid-capture-and-streaming.

[14] L. P-LINK Technologies Co., *Tl-wn722n_ds*, 2010.

[15] Oculus VR, *Oculus rift developer guide - dg.pdf*, 2016. [Online]. Available: http://static.oculus.com/documentation/pdfs/pcsdk/latest/dg.pdf.

[16] O. Wijk, P. Jensfelt, and H. I. Christensen, *Triangulation based fusion of ultrasonic sensor data*, 1998. DOI: 10.1109/ROBOT.1998.680966.

[17] Lightware, *28054-sf11-laser-altimeter-manual-rev-1.pdf*, 2016. [Online]. Available: https://www.parallax.com/sites/default/files/downloads/28054-SF11-Laser-Altimeter-Manual-Rev-1.pdf.

[18] R. E. Berg, *Ultrasonics*, 2016. [Online]. Available: http://academic.eb.com/EBchecked/topic/613488/ultrasonics.

[19] The Engineering Toolbox, *Stp - standard temperature and pressure & ntp - normal temperature and pressure*. [Online]. Available: http://www.engineeringtoolbox.com/stp-standard-ntp-normal-air-d%7B%5C_%7D772.html.

[20] Parallax Inc, *Ping))) ultrasonic distance sensor (#28015) - 28015-ping-sensor-product-guide-v2.0.pdf*, 2013. [Online]. Available: https://www.parallax.com/sites/default/files/downloads/28015-PING-Sensor-Product-Guide-v2.0.pdf.

[21] ElecFreaks, *Ultrasonic ranging module hc - sr04*. [Online]. Available: http://www.micropik.com/PDF/HCSR04.pdf.

[22] Arduino, *Arduino - arduinoboarduno*, 2016. [Online]. Available: https://www.arduino.cc/en/main/arduinoBoardUno.

[23] M. O. McWilliams, *Magnetometer - accessscience from mcgraw-hill education*, 2014. DOI: http://dx.doi.org/10.1036/1097-8542.399600. [Online]. Available: http://www.accessscience.com/content/magnetometer/399600.

[24] B. Gross, *Ultrasonic sensor operation on a quadcopter*, 2013. [Online]. Available: http://www.maxbotix.com/articles/067.htm.

[25] D. P. Havens, *Electric filter - accessscience from mcgraw-hill education*, 2014. DOI: http://dx.doi.org/10.1036/1097-8542.216100. [Online]. Available: http://www.accessscience.com/content/electric-filter/216100.

[26] Arduino, *Arduino - analogread*, 2016. [Online]. Available: https://www.arduino.cc/en/Reference/AnalogRead.

[27] A. C. Frank, *Online calculator .:. active butterworth bandpass filter calculator*, 2014. [Online]. Available: http://www.changpuak.ch/electronics/Butterworth%7B%5C_%7DBandpass%7B%5C_%7Dactive.php.

[28] Ardupilot Dev Team, *Copter home — copter documentation*, 2016. [Online]. Available: http://ardupilot.org/copter/index.html.

[29] ——, *Compatible rc transmitter and receiver systems (pixhawk/px4) — copter documentation*, 2016. [Online]. Available: http://ardupilot.org/copter/docs/common-pixhawk-and-px4-compatible-rc-transmitter-and-receiver-systems.html.

[30] ——, *Flight modes — copter documentation*, 2016. [Online]. Available: http://ardupilot.org/copter/docs/flight-modes.html.

[31] ——, *Code overview (copter) — dev documentation*, 2016. [Online]. Available: http://copter.ardupilot.org/dev/docs/apmcopter-code-overview.html.

[32] V. Toochinda, *Digital pid controllers*, 2014. [Online]. Available: http://www.controlsystemslab.com/doc/b4/pid.pdf.

[33] The Qt Company, *Qt - home*, 2016. [Online]. Available: https://www.qt.io/.

[34] A. B. Coppens, *Sound*, 2014. DOI: http://dx.doi.org/10.1036/1097-8542.637200.

[35] E. Brynjolfsson and A. McAfee, *The second machine age: work, progress, and prosperity in a time of brilliant technologies*. New York: WW Norton & Company, 2014. [Online]. Available: https://www.google.com/books?hl=en%7B%5C&%7Dlr=%7B%5C&%7Did=PMBUAgAAQBAJ%7B%5C&%7Doi=fnd%7B%5C&%7Dpg=PA1%7B%5C&%7Ddq=second+machine+age%7B%5C&%7Dots=y2VN3th6TS%7B%5C&%7Dsig=5Rn16Nlj2Ix8hgvykZ6uxylpMgc.

[36] Raspberry Pi Foundation, *Raspberry pi foundation - about us*. [Online]. Available: https://www.raspberrypi.org/about/.

[37] DJI, *Dji - about us*. [Online]. Available: http://www.dji.com/company.

A

Appendix 1

A.1 Components

Frame	DJI Flame wheel F450
Propeller	4 x DJI 1038 10-inch
Motors	4 x DJI 2212/920KV
ESC	4 x DJI 30A OPTO
Flight controller	Ardupilot APM 2.6
Landing gear	DJI landing gear for flame wheel F450/F550
Battery	Gravity 5500 3S 30-40 C LiPo
GPS	3DR uBlox GPS with Compass Kit
Telemetry	3DR Radio set 433 MHz
Receiver	FrSky V8FR
Ultrasonic sensor	3 x Parrallax PING))), 1 HC-SR04
Single chip computer	Raspberry Pi 2B
Microcontroller	Arduino Uno
Voltage converter	Hobbywing UBEC, 3A

Camera	Raspberry Pi Camera Module
HMD	Oculus Rift DK2

B

Appendix 2

B.1 Source Code

The source code is available as a git repository hosted on Github. It is available here: https://github.com/JMilleson/tredjepersonsvy.git

B.2 piApp

The specification for the interface to the pyApp server. It uses formatted Strings in the form of JSON. The following JSON objects are accepted:

`{`
`  "requestVideo": {`
`  "timeout": t,`
`  "width": w,`
`  "height": h,`
`  "bitrate": b,`
`  "QP": q,`
`  "port": p`
`  }`
`}`
	Attempts to start a video stream using raspivid and pipe it using UDP over the Wi-Fi using gstreamer with the settings specified. It stops after t seconds. It will stream to the ip of the requester on port p.

`{`
`  "abortVideo": ""`
`}`
	This kills the camera immediately and drops all streaming pipelines.

<pre>{
  "requestSensorData": {
  "intervall": x
  }
}</pre>
	Starts sending sensor data to the client who requested it with the interval of about x milliseconds. If no interval is specified it sends sensor data whenever it wants. The sensor data comes from the Pinger.py module and includes the distance between the quadcopter and the user and the centre line off-set.

<pre>{
  "calibrateSensors": ""
}</pre>
	Immediately calibrates the tracking module. This temporary disables the capability of getting distance and centre line off-set. It presumes the user is 300cm from the quadcopter and calibrates accordingly. Calibration is needed before attempting to get the distance between the user and the quadcopter.

<pre>{
  "sensordataoculus": {
  "roll": x,
  "pitch": y,
  "yaw": z,
  }
}</pre>
	This is used to send sensor data describing the rotation of the user. piApp stores the information and sends it to the ARC the next time it is going to send data.

{
"settings": {
"target": {
"targetHeight": x,
"targetDistance": y
},
"pitch": {
"p": x1,
"i": y1,
"d": z1
},
"roll": {
"p": x2,
"i": y2,
"d": z2
},
"yaw": {
"p": x3,
"i": y3,
"d": z3
},
"trottle": {
"p": x4,
"i": y4,
"d": z4
}
}
}
	Tells piApp to send new settings to ARC. Takes settings for the p,i and d parameters for all the PIDs as well as the target height and the target distance for the quadcopter. It sends the settings the next time it is possible to do so.

B.3 Quadcopter Height Calculation

The measured height h0 from the ping sensor can be represented by the length of a vector in the three dimensional room.

$$\vec{v} = x * \hat{x} + y * \hat{y} + z * \hat{z}$$

$$(*)h_0 = |\vec{v}| = \sqrt[2]{x^2 + y^2 + z^2}$$

Where z is the real height acquired after h_0 have been compensated for the pitch angel α_p and the roll angle α_r.

$\vec{v}$ is then projected on the yz plane respectively the xz plane to be able to form two equations with α_p respectively α_r.

$$proj_{yz}(\vec{v}) = 0 * \hat{x} + y * \hat{y} + z * \hat{z} =>$$

$$tan(\alpha_p) = \frac{y}{z} =>$$

$$y = z * tan(\alpha_p)$$

$$proj_{xz}(\vec{v}) = x * \hat{x} + 0 * \hat{y} + z * \hat{z} =>$$

$$tan(\alpha_r) = \frac{x}{z} =>$$

$$x = z * tan(\alpha_r)$$

The two expressions for x and y are then put into the (*) expression.

$$h_0 = \sqrt[2]{z^2 * tan(\alpha_r)^2 + z^2 * tan(\alpha_p)^2 + z^2} =>$$

$$z = \frac{h_0}{\sqrt[2]{tan(\alpha_r)^2 + tan(\alpha_p)^2 + 1}}$$

B.4 Video Transfer

To start the video transfer the following command is used:

```
raspivid -t 00 -w 1080 -h 948 -fps 30 -b 000 -n  -o - |\
gst-launch-1.0 -v fdsrc ! h264parse ! rtph264pay \
config-interval=1 pt=96 ! udpsink host=192.168.42.11\
port=1337
```