Development and Implementation of an Unmanned Aerial Vehicle with Stereoscopic Cameras Controlled via a Virtual Reality Head-Mounted Display

Master Thesis

Faculty: Computer Science and Engineering

Submitted by:	Marcus Scherer
Program:	High Integrity Systems
Matriculation number:	85 46 83
Examiner:	Prof. Dr. Christian Baun
Co-examiner:	Prof. Dr. Matthias F. Wagner

Declaration of Authorship

I, Marcus Scherer, declare that this thesis titled, 'Development and Implementation of an Unmanned Aerial Vehicle with Stereoscopic Cameras Controlled via a Virtual Reality Head-Mounted Display' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Frankfurt am Main, 27th of February, 2015

Marcus Scherer

This thesis is lovingly dedicated to my mother.

Acknowledgement

Since this thesis marks the end of my studies at the Frankfurt University of Applied Science, I would like to thank everyone who helped me to achieve this goal.

In particular I would like to thank:

- Torsten and Alex for working with me through hard lectures and always keeping my spirit high.
- Thomas Hey'l for his continuous support in all areas.
- Gregor Metternich and Diana Mäser for helping me with the medical research.

Abstract

The goal of this thesis is to develop an unmanned aerial vehicle which is capable of providing a stereoscopic overview of a given area. Emergency units and search and rescue parties shall be given an easy access to use the tool, with which they can monitor a site of an accident. The system provides a stereoscopic, tilt and pan able live video feed. Since the rescue team is working under time pressure, the usage of the system should be as easy as moving your head around. This goal is achieved by using a hexacopter that is equipped with two cameras. The video signal of the cameras is transmitted to a ground station and inside a virtual reality headset. The movement of the headset is instantly translated to movement of the cameras. The hexacopter is capable of hovering over a given coordinate or flying along a predetermined path.

Contents

1. Introduction

Drones are gaining more and more popularity. Amazon is developing a drone to deliver small packages directly from the warehouse to the customer[1]. DHL, a german delivery firm, had a prototype drone autonomously delivering pharmaceutical medication from the german main land to the island Juist[2]. The TU Delft developed an ambulance drone[3], which carries a defibrillator.
The fast delivery and the video communication with an emergency operator may increase the chance of a patient to be resuscitated successfully after a cardiac arrest.
In the "Hype Cycle for Emerging Technologies" of 2014 published by Gartner[4], drones or autonomous vehicles are expected to reach the plateau of productivity within the next 5 to 10 years.

In the early 90s virtual reality (VR) and VR headsets were believed to be the next big thing. This never happened, because the technology was not up for this big task, the hardware was too expensive, not enough software existed that took advantage of VR and people just got sick using it (simulator sickness).
But with hardware like the Oculus Rift, VR really seems to become the next big thing. That may be the reason why Facebook bought Oculus for 2 billion USD[5]. The hardware has become as cheap as a regular computer display. The technology has made huge steps in terms of display resolution, sensors and processing power. The reasons behind simulator sickness are understood slowly and can now be fought.

[1] http://www.amazon.com/b?node=8037720011
[2] http://www.dhl.com/en/press/releases/releases_2014/group/dhl_parcelcopter_launches_initial_operations_for_research_purposes.html
[3] http://www.tudelft.nl/en/current/latest-news/article/detail/ambulance-drone-tu-delft-vergroot-overlevingskans-bij-hartstilstand-drastisch/
[4] http://www.gartner.com/newsroom/id/2819918
[5] http://newsroom.fb.com/news/2014/03/facebook-to-acquire-oculus/

1.1. Motivation

Search and rescue (SAR) missions have to find and provide aid to a person in state of emergency or imminent danger as fast as possible. Depending on the search area different tools are used to support the SAR teams, like rescue dogs, boats, planes or helicopters. For a smaller or urban environment planes or helicopters can be to expensive or just to big and not precise enough. A small and rapidly deployable tool that is capable of scanning a given area, would be a great tool for SAR missions.
Rescue units like firefighters and paramedics need to get a quick overview of a crash or disaster site. People who know the site might be under shock and may not be able to give a reliable description of the site. Just as with the SAR a tool that gives a fast and quick overview would be of great use.

1.2. Goal

To help SAR and disaster response units a system will be developed and implemented that can provide a survey over a region. The system must be quickly to deploy and easy to use. A drone is developed to achieve this purpose. It will carry two cameras that will provide a stereoscopic video stream. The drone will be capable to autonomously hover over a given position or fly along a predefined path. The video stream will be displayed to the user with a VR headset. The created three-dimensional video will help to spot a mission person. To provide a more natural way to look around, the head movement will be used to turn the camera. This will create the illusion that the user is positioned below the drone and can look around freely.

1.3. Structure

In the second chapter of this text an overview over different types of drones and the VR headset, Oculus Rift, is given. The rest of the chapter will give examples of drones combined with first person flight or the Oculus Rift. The chapter Design will

go into detail what general components are needed, how they work and which parts have finally been chosen to be used in the system. The last part of this chapter will describe two pieces of software that need to be developed. The forth chapter first describes where which parts are going to be placed. Then the build process of the drone is described. The last two sections of the chapter will go into detail how the two missing applications have been implemented and what their tasks are. After this the developed solution will be evaluated. At the end a conclusion as well as possible enhancements will be given.

2. State of the Art

This chapter contains an overview about unmanned aerial vehicles and the Oculus Rift, a virtual reality headset. After that an overview over existing Drone solutions and those which try to combine Drones with the Oculus Rift is given.

2.1. Unmanned Aerial Vehicle

An unmanned aerial vehicle (UAV) is a flying vehicle, which is capable of autonomous flight or/and is controlled via radio control (R/C). UAVs are often referred to as drones. There are UAVs in a multitude of sizes. In general UAVs can be divided into two categories: fixed-wing- and rotor-designs.

2.1.1. Fixed-wing

Fixed-wing UAVs have the benefit of an increased flight time and range. Those benefits are based on the capability of gliding without losing much height, even without propulsion. This design makes it impossible to hover over a certain position. The only alternative is to circle around a position. Due to the increased range most UAVs at the military are fixed-wing. One of the most known UAVs is the Predator, see Figure 2.1, from the US military. Positional changes are achieved as with a regular airplane, e.g. thrust, rudder, elevators and ailerons.

2.1.2. Multicopters

A multicopter does not have wings and flies with three or more rotors. Most common types are quadcopter, hexacopter and octacopter with four, six or eight rotors. To increase or decrease height all rotors are given more or less amounts of thrust.

Figure 2.1.: MQ-1 Predator Source: `http://commons.wikimedia.org/wiki/File:MQ-1_Predator_unmanned_aircraft.jpg`

2.2. Oculus Rift

The Oculus Rift is a virtual reality head-mounted display from the company Oculus VR. Currently the Rift is still in development and no consumer version is available. From end of 2012 till March 2014 the Development Kit 1 (DK1) was available for order for interested developers. Since March 2014 the Development Kit 2 (DK2) can be ordered. The DK2 features an organic light-emitting diode (OLED) display with a resolution of 1920x1080 pixels. It is the same display used as by the Samsung Galaxy Note 3 smartphone (sam). It has a refresh rate of 75 Hertz (Hz) with a low persistence of 2-3ms. To see the screen sharp so close to the eyes two lenses are mounted in front of the display and provide a 100° field of view. The Rift is capable of tracking the head movement and positional movement of the user. Head movement is tracked with a gyroscope, an accelerometer and magnetometer with an update rate of 1,000 Hz. For the positional tracking the Rift has 40 infrared light-emitting diodes (LED) on the headset and an infrared camera which needs to be placed in front of the user on eye height. The camera and the Rift are connected to the personal computer (PC) via Universal Serial Bus (USB). The Rift gets also connected to an High-Definition Multimedia Interface (HDMI) port of the PC.

2.2.1. Simulator Sickness

Simulator sickness, also called motion or gaming sickness, can cause among other symptoms dizziness, headache, double vision and nausea (AGS12) (MPB14). It can be caused by a simulation that presents the user an image that clashes with the users daily experience, e.g. wrong physics, distorted vision or images that seem delayed (MT14). Another cause of simulator sickness can be a discrepancy between the vestibular system and the images presented (TK07). A simulated situation that would normally create a force on the body, like the acceleration in a car, is an example. The user perceives that the car is getting faster and that he is sitting inside but he is not felling any g-forces on its body since he is sitting in a regular chair with a VR-Headset on his head.

2.2.2. Distortion

The lenses used in the Rift are creating a so called Pincushion Distortion. To counter this effect the software providing the images for the Rift needs to apply a Barrel Distortion. Those both Distortions will cancel each other out and the user will see a not distorted image (rif).

2.3. Existing Solutions

This part presents an overview over already existing commercial systems or prototypes and compares their feature sets with the needed features of this project.

2.3.1. DJI Phantom 2

The DJI Phantom 2 (Figure 2.2) is one of the most often used drones (JN) in amateur drone flying and areal filming. The design of the drone is a quadcopter with an existing mounting point for a camera gimbal and a mini-USB port for power delivery. The autopilot is capable of hovering over a given position and can

Figure 2.2.: DJI Phantom 2, Source: http://wiki.dji.com/en/index.php/Phantom_2

be programmed to fly along a predefined path. To use those features the "DJI 2.4GHz Datalink" is needed, which enables the Phantom 2 to be controlled via a Ground Control Station (GCS). Since the Datalink uses 2.4GHz Bluetooth the range is limited to under 1.1km. (dji)

2.3.2. Oculus FPV

The Oculus FPV[6] (EH14) uses the DJI Phantom 2 (2.3.1) as a platform. To the bottom of the UAV a plywood construction with two cameras and two servos has been attached. A servo is an actuator that allows precise and controlled positioning of its arm. The servos are moving the plywood construction and therefore the cameras. Servocommands are transmitted via a 433MHz wireless serial link with a reach of under 500meters. Video signals are transmitted via two 5.8GHz videolinks and converted with two USB video converters, which feed the video signal into a

[6] https://github.com/Matsemann/oculus-fpv

computer. The computer projects the videofeed inside a Oculus Rift, head movement is used to steer the servos, which are moving the cameras.

2.3.3. Intuitive Aerial

Intuitive Aerial, a swedish drone manufacturer, used one of there multicopters and attached a wooden box to it[7]. Inside the box is a laptop with two USB-webcams. The laptop streams the videofeed of the two cameras via WiFi to another laptop on the ground. An Oculus Rift is connected with the laptop on ground showing the streamed video feed. Due to the WiFi transmission the delay in the videofeed is around 120ms. Since the wooden box is not able to move, head movement of the Oculus Rift is ignored in this prototype. The range of the transmission is not mentioned, but since WiFi is used, it should be limited.

2.3.4. Parrot Bebop

The US-based Company Parrot introduced the "Bebop Drone with Sky Controller"[8] during the CES[9] 2015. This package includes two parts, the drone itself and a remote control device.

Bebop Drone

The drone[10] itself is a quadcopter with a single 14 Megapixel videocamera with a 180° field of vision. It is controlled via a 2.4GHz or 5GHz WiFi connection with a range of 250 meters.

[7] http://hackaday.com/2013/07/16/fpv-drones-with-an-oculus-rift/
[8] http://www.parrot.com/usa/products/skycontroller/
[9] International Consumer Electronics Show - an annual fair for consumer electronics
[10] http://www.parrot.com/usa/products/bebop-drone/

Sky Controller

The Sky Controller features joysticks to control the drone. With the integrated signal booster the range of the WiFi connection is increased to up to 2 kilometers. Tablets can be mounted directly to the Sky Controller and can show a live videofeed or be used to program the drone. A HDMI output is also available, which can be used for video glasses or the Oculus Rift. From the product page it is hard to determine if the Oculus Rift is capable of controlling the camera or the drone. A USB-Port is available for "several usage (sensors)", which could be connected to the Oculus Rift for polling the sensors for head movement data.

3. Design

This chapter will first give an overview of the required features and a brief listing of the needed components to fulfill the features. Afterwards the task of each component and its ideal feature set is described. Finally the used components and their features will be described.

3.1. Requirements

To achieve the set of goals a UAV needs to be build which is capable of hovering over a given area, can fly along a given path, has two moveable cameras, wireless video transmission and can take commands from a GCS. The GCS must be capable to receive and display the stereoscopic video. It also needs to monitor the head movement of the user, generate commands to move the cameras accordingly and send those commands to the UAV.

3.2. Flight Management Unit

The task of the Flight Management Unit (FMU) is to keep the UAV stable in flight and provide auto pilot features for the UAV.

3.2.1. Input/Output Modules

Since the FMU does not have any output/input to get data, like the GPS-Position (Global Positioning System), or to control the engines, a module is needed, which is capable of those tasks.

3.2.2. Pixhawk

Pixhawk[11] is an open-hardware platform, which combines the FMU with an input/output module. It includes some sensors, which provide vital data, to achieve a stable flight. It features among other things:

- A 168MHz ARM Cortex-M4 CPU
- 14 × Servo outputs
- 16bit Gyroscope
- 14bit Accelerometer/Magnetometer
- Barometer
- 5 × Serial Ports (UART)
- MicroUSB-Port
- MicroSD card slot

This flight controller is chosen over others, like the Naza-M V2, because it is open-Hardware, which means that every internal detail of the hardware is open to the public[12]. The CPU leaves enough headroom to run additional programs on the FMU. Another benefit of the Pixhawk over other flight controllers is the option to choose the flight stack.

3.3. Flight Stack

In contrast to the Flight Controller the flight stack is software which runs on the FMU. The software needs to process all given data from the sensors (gyroscope, GPS, accelerometer, magnetometer) to get an image of its position in the three-dimensional space. The flight stack reacts to given inputs to adjust its position. The input can come from a remote control with basic commands, like increase/decrease

[11] `https://pixhawk.org/modules/pixhawk`
[12] `https://github.com/PX4/Hardware`

throttle or roll. The flight stack also has to handle more complex commands, like to stay on the same spot, despite hard winds or to move to a GPS-Position. As mentioned before the Pixhawk supports a multitude of flight stacks. The two most widely used are: PX4 and APM.

3.3.1. PX4

PX4 has been chosen while testing the development of this project. The following observations have been made:

- it is the most stable one
- it is the one which is best documented
- it has the best performance

The inferior performance of APM can be attributed to the fact that APM builds and runs on top of PX4.

This flight stack uses NuttX, a real-time operating system (RTOS), which provides a POSIX-style environment. Among other things, NuttX provides basic file operations like open() and close() and threads with prioritizations.

Flight modes

The six flight modes of the PX4 can be divided into three categories, see Figure 3.1. The manual mode leaves total control to the R/C and the autopilot does not intervene.
The modes in the assisted category increase the amount how much the autopilot intervenes. The ALTCTL (altitude controlled) mode control inputs are handled just like in the Manual mode, except of thrust, which only indicates climb and sink. The POSCTL (position controlled) mode gives more control to the autopilot then the ALTCTL mode. Roll and pitch determine only how fast the UAV should move in the given direction.

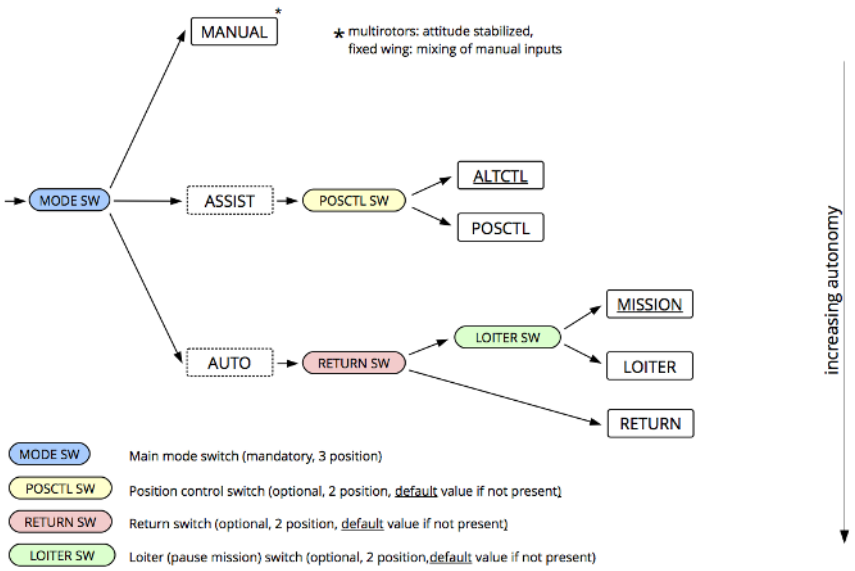

Figure 3.1.: PX4 Flight Modes Source: `https://pixhawk.org/users/system_modes`

The last three modes in the Auto category ignore any commands from the R/C. The RETURN/RTL (return to launch) mode prompts the autopilot to return to the launch-position and land there. The LOITER mode holds the UAV at the current position and height. In the MISSION mode the autopilot works off all mission items, which have been set via the GCS. If no item is programmed to the autopilot, the autopilot will fallback to the LOITER mode.

3.4. Ground Control Station

A GCS is a ground-based control station, which provides utilities for humans to command and control one or multiple UAVs. In this project the GCS is used to set mission items, track the UAV and set the flight mode of the UAV.

3.4.1. QGroundControl

QGroundControl is a open-source GCS with support for Linux®, Mac OS® and Windows®. It supports Micro Air Vehicle Link (MAVLink) and can send and receive MAVLink messages over User Datagram Protocol (UDP), serial or mesh networks. Besides other autopilots Pixhawk is fully supported. Position and other telemetry data can be tracked and plotted in real-time. Waypoints and flight mode of the Pixhawk can be set with QGroundControl during flight. Critical warnings, issued via MAVLink or detected by QGroundControl, like a low battery, can be read out by QGroundControl. In Figure 3.2 QGroundControl can be seen connected to a landed UAV.

Figure 3.2.: Screenshot: QGroundControl

3.5. MAVLink

MAVLink is a communication protocol. It is designed to be fast in transmission and safe against transmission errors. It was first released in 2009 by Lorenz Meier under the LGPL license. MAVLink is used for the communication between the GCS and UAV.

3.5.1. MAVLink Package Structure

A MAVLink frame can be between 8 and 263 bytes long. The first byte is set to 0xFE and indicates the beginning of a new packet. Payload length is indicated with the second byte and can vary from 0 to 255 bytes. For packet loss detection the third byte is used to count up a packet sequence. Every component has to count up this value when sending a new packet. If a receiver gets a packet with a sequence number higher than expected, the previous can be assumed as lost. Packet sequence counts up from 0 to 255 and starts then at 0 again. Byte 4 and 5 mark the sending system and component. Typ of the payload is indicated with the sixth byte. A list of all available payload/message types can be found online under: `https://pixhawk.ethz.ch/mavlink/`. The last two bytes are low and high byte of a hash generated over the whole packet expect the first byte and are used as a checksum. As hash algorithm CRC-16-CCITT is used, which is also used by the SAE Aerospace Standard AS5669a (SAE14).

3.5.2. MAVLink Messages

MAVLink messages are used for basic task like, transmitting information about the UAV(`SYS_STATUS`), setting the flight mode(`SET_MODE`) or setting new waypoints (`MISSION_ITEM`). `COMMAND_LONG`[13] message is somewhat different, since it offers a command ID and seven parameters to be set. This allows to have more then 255 messages to be defined, since the command ID is a 16-bit unsigned integer.

[13] `http://mavlink.org/messages/common#COMMAND_LONG`

MAVLink Commands

A list of the available MAVLink commands can be found here: `http://mavlink.org/messages/common`. Exemplary commands would be, `MAV_CMD_DO_SET_HOME` for setting a new home location or `MAV_CMD_DO_PARACHUTE` to trigger a parachute. One command is very useful for this project: `MAV_CMD_DO_SET_SERVO`. This allows to set the pulse-width modulation (PWM) value for a given servo. The first parameter of MAVLink message `COMMAND_LONG`, determines the servo number and the second parameter the PWM value.

3.6. Radio Data Link

Communication between the UAV and GCS is wireless. For this purpose two transmitters are needed. One is connected to the UAV and one to the GCS.

3.6.1. 3DR Radio Set

The 3DR Radio Set[14] transmits over the 433MHz frequency. Each module can either be connected to Pixhawk via a 6-wire cable or via a micro-USB cable to a computer.

3.7. MAVProxy

MAVProxy[15] is a GCS written in python[16] with great flexibility in multiplexing and forwarding MAVLink messages via USB, serial or network. Those features are needed to use QGroundControl at the same time as Skynet since there is only one wireless connection to the UAV.

[14] `https://store.3drobotics.com/products/3dr-radio`
[15] `http://tridge.github.io/MAVProxy/`
[16] Python - an object-oriented programming language

3.8. Oculus Rift

The Oculus Rift is used to display the stereoscopic video from the UAV. The data from sensors which track the head movement are used to move the cameras.

3.9. Cameras

The cameras need a wide field of vision, to imitate the human eye. Without moving a single human eye has a field of view with 95° away from the nose, 60° towards the nose, 75° downwards and 60° upwards (RK10) (SLH11). A camera with a field of view with 95° would be ideal. To provide good vision, even in low light situations, like a cloudy sky, dusk or dawn, the cameras have to be very light sensitive. To keep the total weight of the UAV as low as possible and because of the limited space, the cameras should be small in size and light in weight. To help with this restriction they can rely on an external power supply and do not need an integrated battery.

3.9.1. Used Cameras

The "600TVL FPV Tuned CMOS" from Fat Shark has been chosen because its features fulfill all the requirements. It is 21x21x12mm in size (without the lens), needs a 3,5-5V power supply and it features a 100° diagonal field of view. The specifications[17] do not mention how light sensitive the camera is, but list "FPV tuned light handling" as a feature. Tests in low light situations have shown that the image quality drops as it gets darker. The produced image quality is still good enough to get a clear picture though.

[17] http://fatshark.com/uploads/pdf/1745-1.pdf

3.10. Camera Mount and Servos

A small and light construction is needed to mount the two cameras to the UAV and move them in pan and tilt direction via two servos.

3.10.1. Pan

Pan is camera movement along the y-axis, corresponding to the movement of a human head from left to right and vice versa.

3.10.2. Tilt

Tilt is camera movement along the x-axis, corresponding to a nodding movement of the human head.

3.10.3. Servos

Since the servos are responsible for actually moving the cameras, the movement radius of the servo is the limiting factor of the degrees of freedom in pan and tilt. Even while sitting, a human is capable of looking in all directions along the y-axis, which would translate to a servo moving 360° for pan. If the upper body is not allowed to move the degrees of head movement is halved to 180°.
The tilt servo needs not to move that much, since a human being can only look up and down covering an angle of nearly 180°. This means that the tilt servo needs to move 180°.

Servo Control

There are three wires attached to a servo, two for direct current (DC) and one for the control signal. The control signal is encoded with PWM. For R/C servos a 50-Hz signal (20 milliseconds (ms) period) is well-established. During this period the servo

expects one pulse and the length of the pulse determines the position of the servo arm. Most servos interpret a 1 ms width pulse as their lowest position and a 2 ms width pulse as their highest position (Sch14). The PWM values used in this thesis are expressed in microseconds. A PWM value of 1,000 is equal to a pulse width of 1ms.

The common servo is capable of moving its arm in a 90° angle, which is just the half the degrees needed for tilt and only a quarter the degrees needed for pan. For the pan movement a Futaba S3003 servo is used. Due to the construction of the camera mount this servo needs to have a higher torque then the tilt servo. Details to the construction of the camera mount can be found later in section 4.2.1. If this servo is controlled with PWM values between 1,000 and 2,000, it is capable of moving approximately by 90° only. But it is capable of being controlled with much lower PWM values which makes it possible to move the servo approximately by 180°. For the tilt movement a Hyperion Atlas DS11-SCB Servo is used. This smaller servo has only half the torque, but it is capable of being programmed. One of the programmable values is the number of the degrees the servo can turn. It can be programmed to a maximum of 175°, which is very close to the needed 180°.

3.11. Video Signal Transmission

The video signal transmission needs to be on a frequency band, which is not used by the remote control (2.4GHz) or the Datalink (433MHz) and should be legal in Germany. It needs to support multiple channels, because each camera needs its own transmission.

3.11.1. Receiver

Two Uno5800 5.8GHz A/V Rx from ImmersionRC are used to receive the video transmissions from the UAV. The receiver can work on seven different channels,

ranging for 5740MHz to 5860MHz with 20MHz for each channel. Channel one and seven are used, to reduce interference.

3.11.2. Transmitter

Two 25mW 5.8GHz A/V Tx from ImmersionRC are mounted to the UAV for the transmission. They can transmit on the same frequencies as the receiver and are setup to use the same channels (1 and 7). They have a transmission power of 25mW. 6 to 25 Volts are supported as supply, which allows direct connection to the battery without intermediate regulators. For video input a 5-pin molex connector is used, that also provides 5 Volt for the camera.

3.12. USB Video Converter

The video signal receivers are providing the two analog video signals via cinch connectors, therefore a converter is needed to provide a digital video stream. The video signal should be converted from analog to digital as late and as fast as possible to reduce latency. Two LogiLink VG0011 video converters are used. The video converter uses the "Syntek STK1160" chipset, which is supported over all major Operating Systems(Windows®, Mac OS®, Linux®).

3.13. AirFrame

The AirFrame is used to house all needed electronics, like the autopilot, the cameras and so forth. As mentioned in 2.1.1 a fixed-wing Airframe would make it impossible to hover stable over a position. That's why a multicopter has been chosen, although its flight time is shorter. In case of an engine failure a hexacopter and octocopter provide more maneuverability. To decrease flight time not more then needed a hexacopter would be the ideal AirFrame in reference to stability, security against engine failure and flight time.

3.13.1. DJI Flame Wheel F550 Set

The DJI Flame Wheel F550 Set contains:

- 6 × 10-inch Propellers
- 6 × Brushless DC electric motors
- 6 × Brushless Electronic Speed Controls (ESC)
- 6 × Arms
- 2 × Centerplates

The assembled AirFrame weights 487 gram and can lift up to 2 kilogram of cargo. The diagonal wheelbase is 550 millimeters and the 200 millimeters wide centerplate provide enough space for the 81.5 × 50 millimeter Pixhawk.

3.14. Remote Control

A remote control is needed to directly control the UAV. A button on the remote control is programmed to put the PX4 into manual mode. This provides a fallback option to control and land the UAV in a case of emergency. The DX6i remote control from Spektrum is used. It features six channels which are transmitted at 2.4GHz frequency. Four of the six channels are used for thrust, roll, pitch and yaw. The fifth channel is used to switch the UAV back to manual mode. Since the 2.4GHz frequency band is used it is not recommended to use the UAV near to residential or office zones or anywhere else with a lot of WiFi signals, since WiFi also uses the same frequency band. The receiver could interpret a WiFi signal as a control command, which could lead to lose of control and in worst case cause a crash.

3.15. Flight Battery

As a power supply for the UAV and all the added video equipment a flight battery is needed. A 3S lithium polymer (LiPo) battery is used for providing power. 3S identifies the number of used cells of the battery, in this case three cells. The three combined cells provide 11,1 Volts.

3.16. Missing Software

For the ground station an application has been developed which is capable of:

- Displaying the stereoscopic video.
- Tracking the head movement.
- Generating MAVLink messages that move the servos according to the head movement.
- Transmitting the MAVLink messages to MAVProxy.

Due to the lack of support for the MAVLink Command "`MAV_CMD_DO_SET_SERVO`" of the PX4 an onboard-application for the PX4 has been developed, which accepts the command and moves servos accordingly.
The development of those programs is described and explained in the next chapter.

4. Implementation

This Chapter describes the implementation of the individual parts of the UAV and the GCS and how they work together.

4.1. Overview

The system can be divided in two sections. The first part contains everything, what stays on the ground and is connected to the computer. This includes:

- 2 × Receiver A/V 5,8 GHz UNO5800 ImmersionRC
- 2 × Video Converter
- 1 × 3DR Radio Telemetrie
- 1 × Computer
- 1 × Oculus Rift
- 1 × MAVProxy (Software)
- 1 × Skynet (Software)
- 1 × GCS (QGroundControl (Software))

These components can be found on the right side in Figure 4.1.

The second part of the system is air bound and either a vital part of the drone or directly mounted to it. The parts mounted to the Hexacopter are:

- 2 × Fatshark 600TVL FPV Tuned CMOS Camera
- 2 × Sender A/V 5,8 GHz 25mW ImmersionRC
- 1 × Camera Mount

- 2 × Servos for Camera Movement
- 1 × 3DR Radio
- 1 × DC Voltage Regulator

These components can be found on the left side in Figure 4.1. Since the remote control is only used as a backup it is not depicted in this overview.

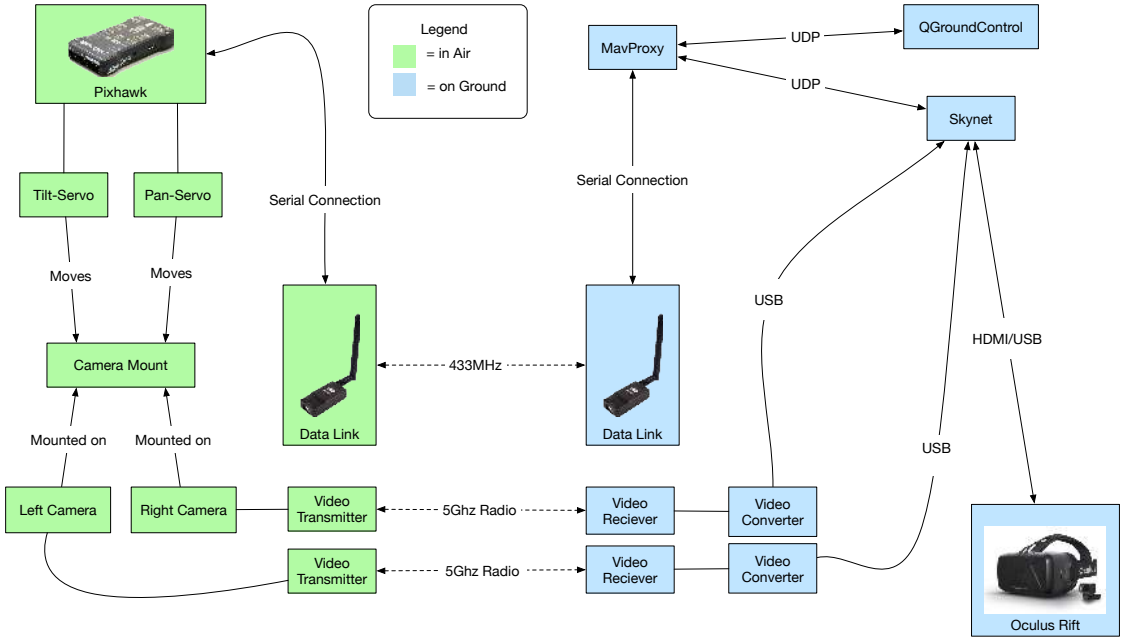

Figure 4.1.: System Overview

4.2. Building the Hexacopter

Construction of the hexacopter starts with the lower centerplate. Power for the ESCs and motors is distributed via the lower centerplate which also functions as a printed circuit board (PCB). The power supplies of the ESCs are soldered to the provided soldering points on the PCB. The power supply which provides power to the PCB is also soldered to the PCB. The soldered PCB can be seen in Figure 4.2. Each of the 6 arms and the four legs of the landing gear get screwed on to the lower centerplate with two M2.5x6 screws. On to the bottom of the arms the ESCs are

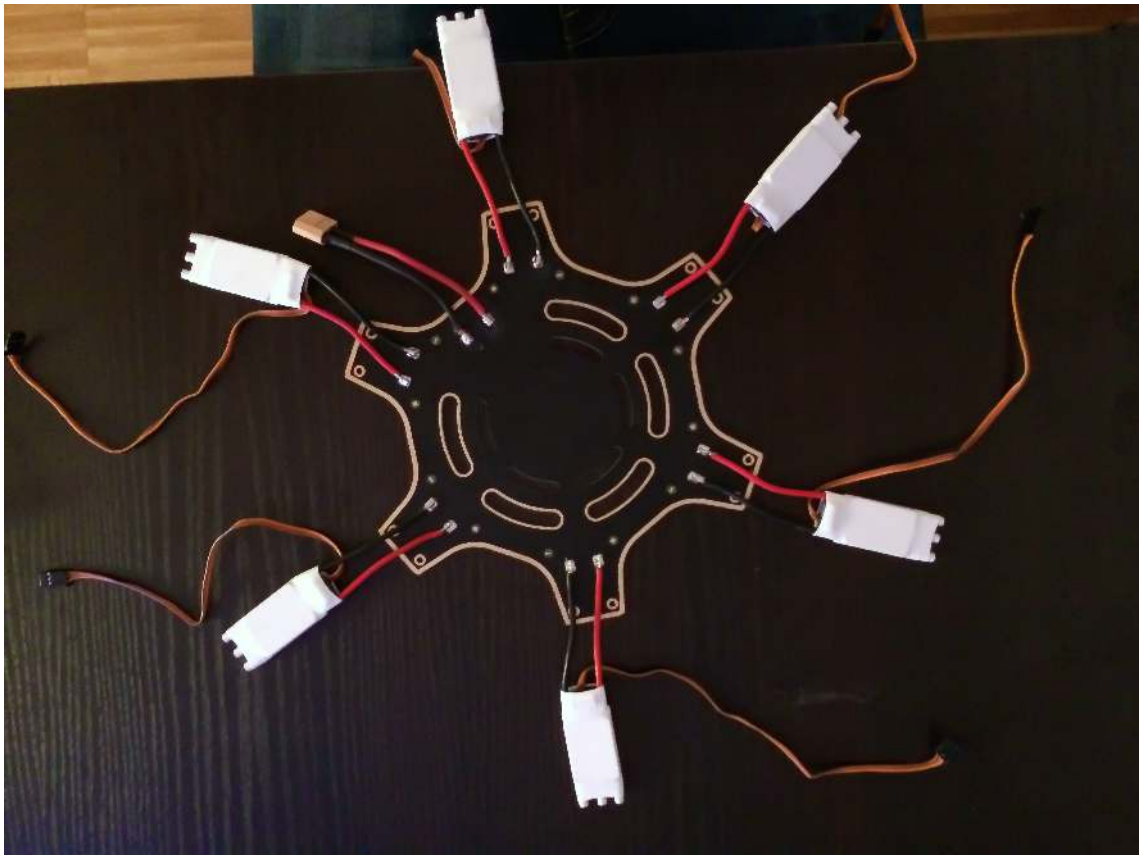

Figure 4.2.: Centerplate with soldered electronic speed controls and power supply.

fastened with two zip ties. The six motors are attached to the end of the arms with four M3x8 screws. The three wires coming from each motor are plugged into the ESC on the same arm as the motor.
Into the middle of the lower centerplate the Pixhawk is placed with the orientation arrow pointing between the two red arms to the front of the hexacopter. This mounting position is ideal, for sensors used by the pixhawk, since it is in the center of gravity of the hexacopter. It is connected to the centerplate with four double-sided adhesive vibration damping pads.
Signal wires from the ESCs are connected to the main outputs 1-6 corresponding to the motor number to which the ESC is connected. Motor numbers can be found in Figure 4.3 for the Hexa X configuration. The cable of the safety switch gets connected to the Pixhawk and the switch is attached with a zip tie to a leg of the landing gear. The buzzer has been fixed at the bottom of the lower centerplate

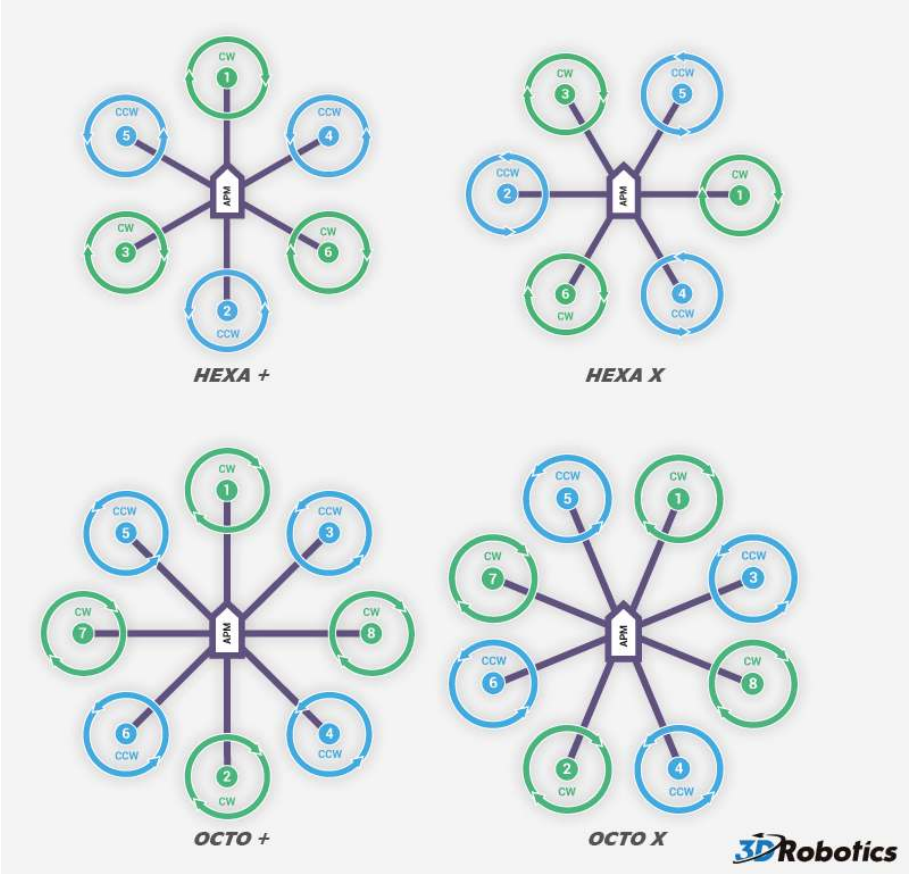

Figure 4.3.: Motor numbers for hexa-and octocopters in + and X configuration Source: `http://copter.ardupilot.com/wiki/initial-setup/motor-setup/`

with double-sided adhesive tape, its cable is connected to the Pixhawk. The Power Module is connected to the Pixhawk and the power supply of the lower centerplate. The receiver for the remote control is glued on top of one arm and connected to the Pixhawk. With a 6-wire cable the 3DR Radio is connected to the Pixhawk and fixed at the bottom of the lower centerplate with double-sided adhesive tape. The antenna of the 3DR Radio is pointing downwards to increase the transmission range.

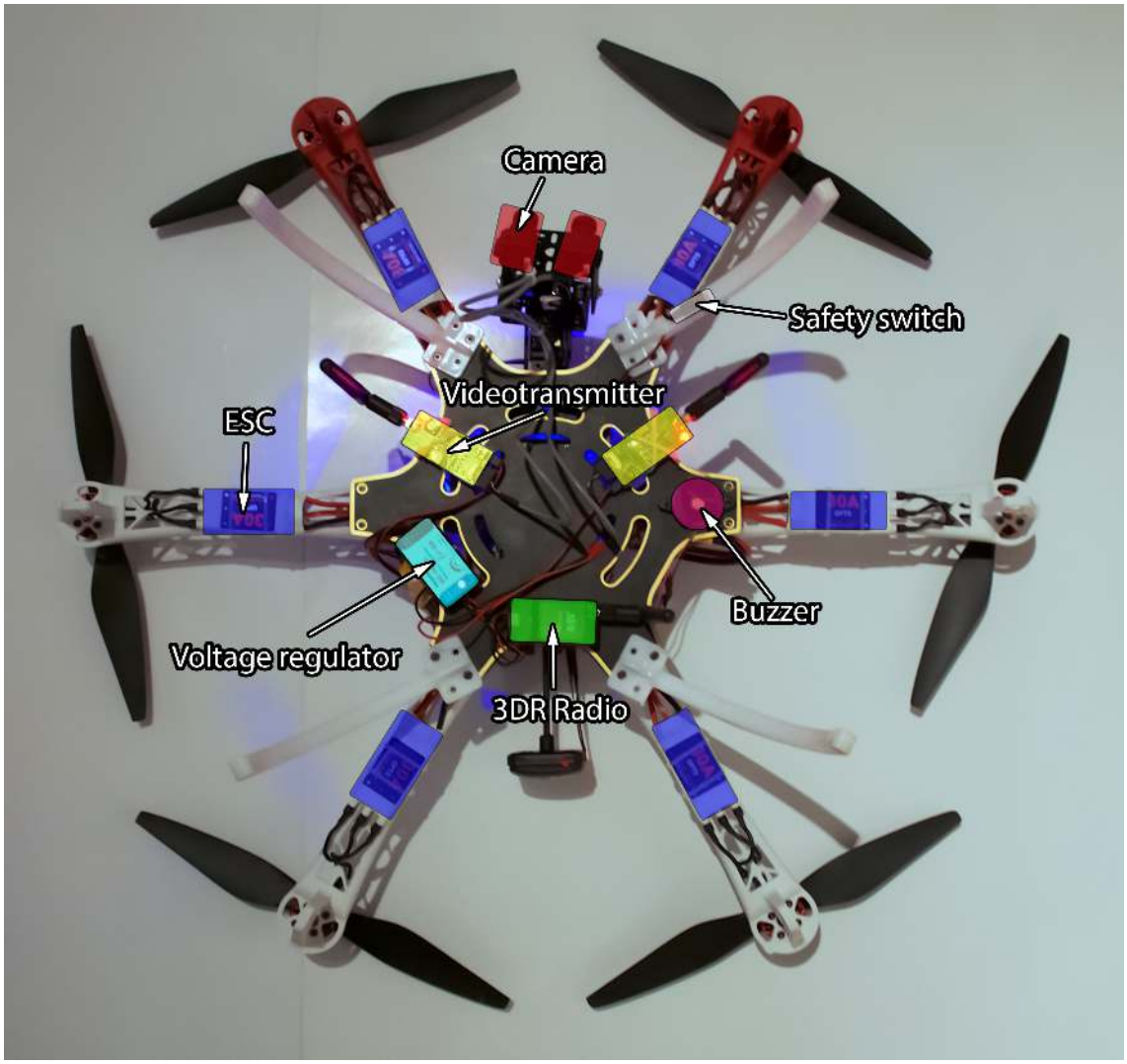

Figure 4.4.: Assembled hexacopter from below

On top of the six arms the upper centerplate is screwed on with four M2.5x6 screws for each arm. The Pixhawk does not contain a GPS but allows a GPS and Compass to be connected. To reduce interference of the GPS and Compass with the electronics of the hexacopter a mast is mounted on top of the upper centerplate. The GPS with integrated compass is mounted on top of the mast and its cables are routed through gaps in the upper centerplate down to the pixhawk, where they are connected. In the middle of the upper centerplate the flight batterie is mounted with hook-and-loop fastener, for easy replacement. The batterie connects to a splitting power cable,

where one end is connected to the power module of the Pixhawk. The other end is connected to a DC voltage regulator and the two video transmitters. The DC voltage regulator is mounted on the button of the lower centerplate and feeds a 5 Volt current to the servo rail of the Pixhawk. The two video transmitters are mounted to the bottom of the lower centerplate in such way to maximize the distance between each other and the 3DR Radio to minimize interference. Each video transmitter is connected to one camera via a single video cable. To allow for movement of the cameras the cables are strapped loosely to the camera mount. The camera mount is attached with zip ties to the bottom of the upper centerplate and sits in front of the hexacopter between the two plates. For better understanding the assembled hexacopter can be seen Figure 4.4 and 4.5 from top and below.

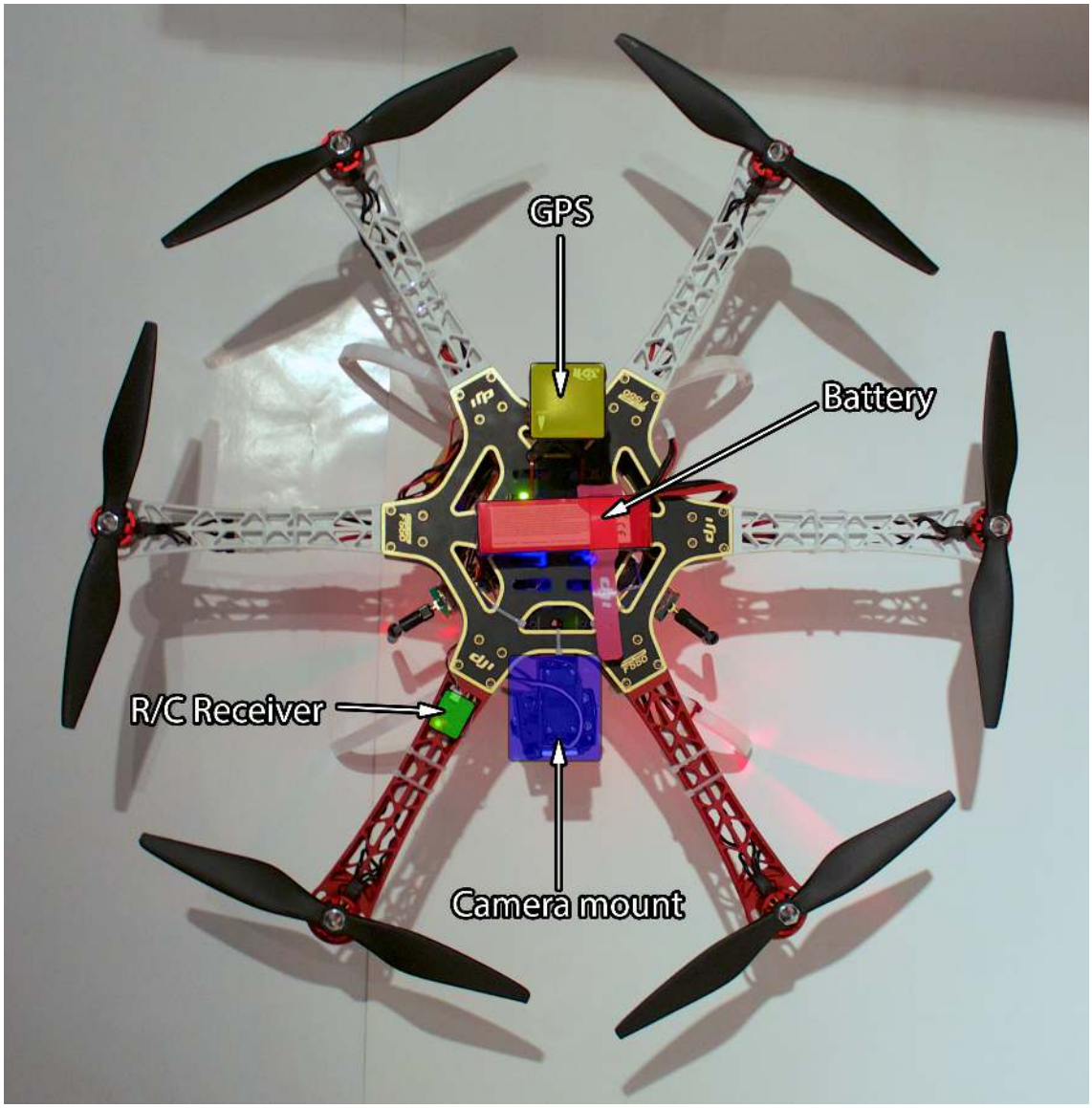

Figure 4.5.: Assembled hexacopter from top

4.2.1. Camera Mount

The camera mount is a construction which is a result of two different commercially available camera mounts. The first one was made out of fiberglass, which helped keeping the weight down. Unfortunately the construction it self was very unstable. Only the part which connected the pan servo to the UAV could be reused for the

final camera mount. On top of the pan servo arm a much stabler and smaller metal construction has been added. This smaller construction houses the tilt servo und provides movement along the tilt axle. Both cameras are superglued to the bottom of the metal construction. The completely assembled and to the UAV attached camera mount can be seen in Figure 4.6.

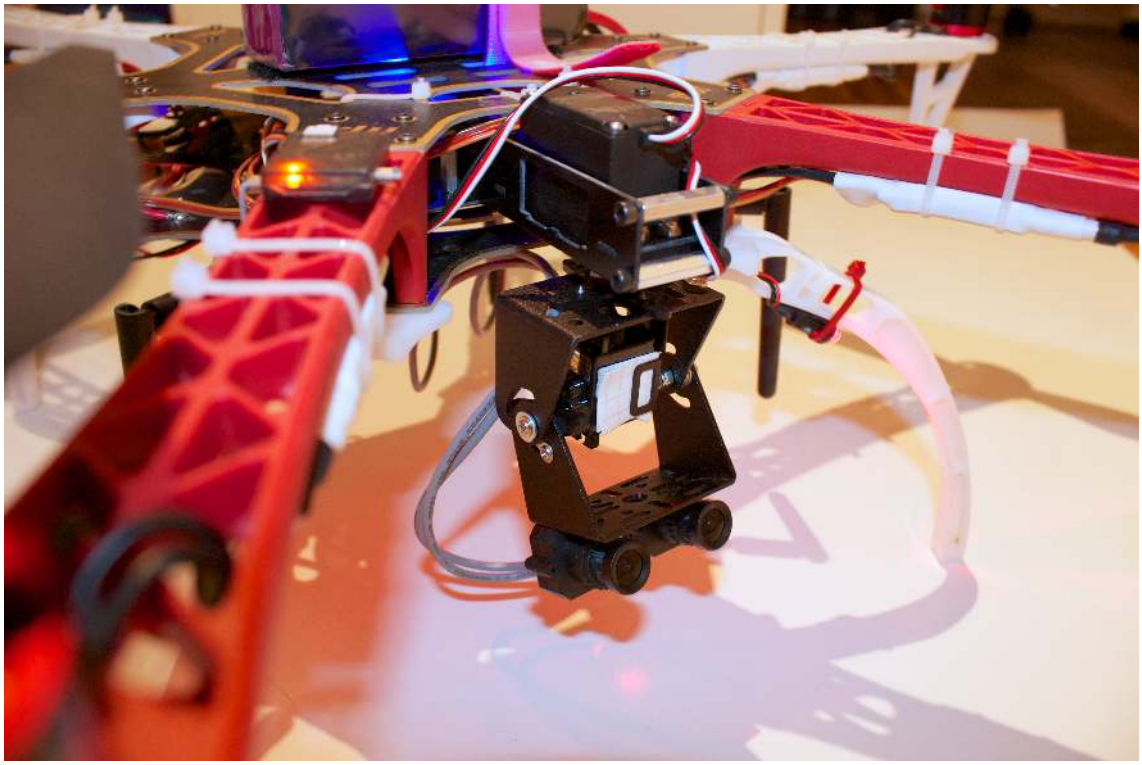

Figure 4.6.: Camera mount

Camera Distance

To create a stereoscopic video the cameras need to be place in the right distance to each other. This distance is called interaxial separation and describes the distance between the centers of two camera lenses. The interaxial separation determines the minimal distance and object can be seen stereoscopic. The 1/30th rule is a rule of thumb to determine the interaxial separation. To calculate the interaxial separation the minimal distance to an object is divided by 30. The result equals the interaxial

separation. The cameras have been mounted with a interaxial separation of 3cm, which equals a minimal object distance of 90cm.

4.2.2. PX4 Configuration

Multiple files of the SD card can be executed during system startup. Two files will be used to change the behavior of the PX4 flight stack.

config.txt

It is possible to override environment variables used by the PX4 with entries in the config.txt-file. The used config.txt can be found in Listing 1. In particular the minimal and maximum PWM values for the rotors as well as the PWM value for disarming are set. The value for the disarmed state needs to be chosen in a way that the rotors won't spin at this value. The minimum value spins up the rotors, but not fast enough to generate any lift. The maximum value gives the value for maximum thrust.

```
set PWM_DISARMED 900    # Motors should stop at this PWM value

set PWM_MIN 1230        # Motors should spin at this PWM value

set PWM_MAX 1950        # Motors should spin with max speed at
 ↪  this PWM value
```

Listing 1: config.txt for PX4

extras.txt

The extra.txt is used to start applications after the system start and to change the behavior of the PX4. After the system start the content of extras.txt is executed in a shell environment. The Listing: 2 shows the extras.txt. First the FMU gets told to

enable the auxiliary outputs on the Pixhawk. These outputs are needed to control the servos which move the cameras. Secondly the onboard application "skynet" is started.

```
fmu mode_pwm        # Tell the fmu to enable auxiliary outputs

skynet start        # Start onboard application skynet
```

Listing 2: extras.txt for PX4

4.3. Ground Station

The main component of the ground station is the computer to which all other parts are connected and on which the used software (MAVProxy, QGroundControl and Skynet) is running. All parts are connected via USB so a USB hub is required due to the lack of enough USB ports of the used notebook. The two video signal receivers transmit their video signals with a chinch cable to the USB video converters. Each converter is then plugged into a USB port. The 3DR Radio is also plugged into a USB port. The Oculus Rift is connected to the computer via HDMI and USB. The camera used for the positional tracking is plugged into a USB port and directly connected to the Rift with a sync cable. The fully assembled ground station is depicted in Figure 4.7.

4.4. Datalink

The transmission of the MAVLink messages over the 433MHz radio link did emerge as a bottleneck for the responsiveness from head movement to camera movement. During the testing phase the communication with the autopilot was done via an USB interface, which had a very small delay, which resulted in a fast reaction of the camera servos. With the 3DR Radio responsible for the communication between autopilot and ground station, a very noticeable and disturbing delay was introduced

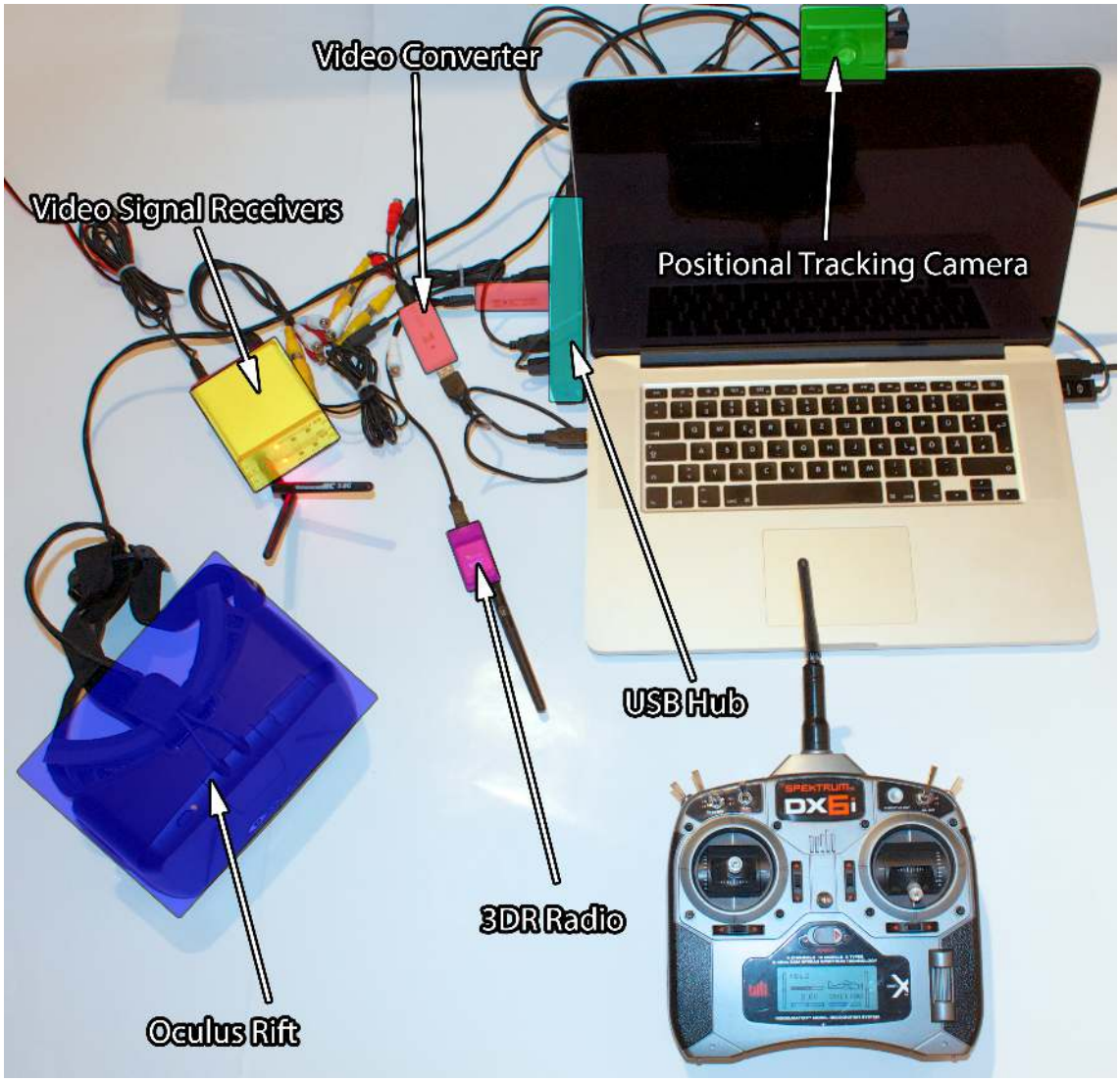

Figure 4.7.: Hardware on the Ground

to the system. In the default configuration the Radiolink is configured for normal payload and an transmission interval of 133ms. Optimistic resend and error correction are also enabled. Figure 4.8 shows the configuration program for the 3DR Radio Set with the old and the new configuration.

To increase the responsiveness the transmission interval has been decreased to 33ms, which guarantees that at least every 33ms a packet will be transmitted. Optimistic resend has been disabled, while it was enabled every packet would get transmitted twice where the second transmission would be marked as a resend. This did lead to twice the amount of data to be transferred. Error correction has also been disabled since it doubles the transmitted data as well and adds 40ms of delay to the decoding process for each transmission.

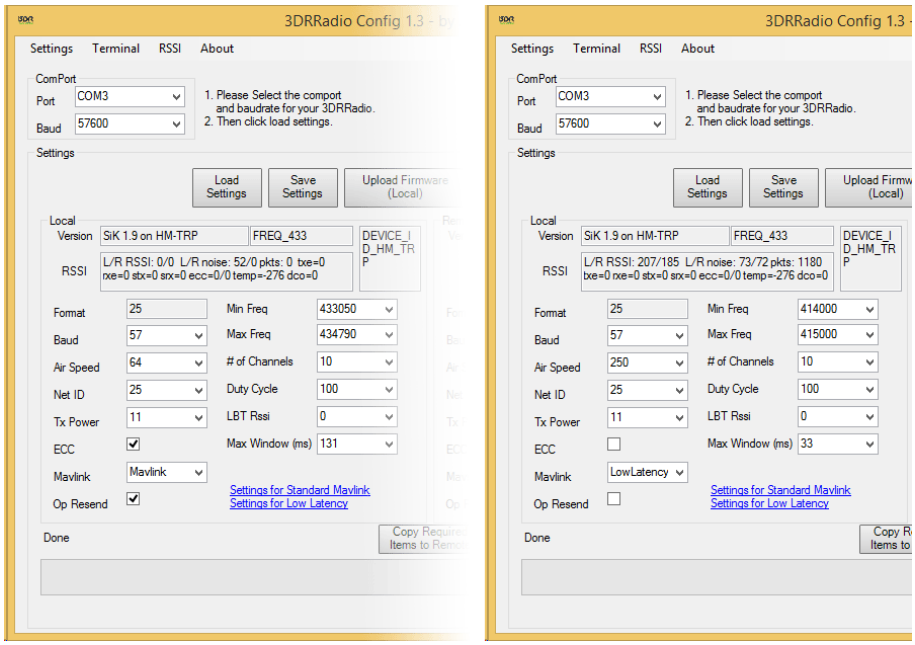

Figure 4.8.: Old Radio Config vs. New Radio Config

4.5. MAVProxy

MAVProxy is configured to connect to the UAV via a USB-serial interface. It is also forwards and accepts MAVLink packets from QGroundControl via UDP on

port 14550, as well on port 31337 for Skynet. Other options that get passed to MAVProxy are the baud rate of the 3DR Radio and the kind of UAV which is connected.

```
mavproxy.py --master=/dev/tty.usbserial-DN00764N --baud=5760
↪ --quadcopter --dialect=pixhawk --load-module px4
↪ --out=127.0.0.1:14550 --out=127.0.0.1:31337
```

4.6. Skynet - Onboard Application

The task of this onboard application for the PX4 flight stack is to parse incoming MAVLink messages with the MAVLink command "`MAV_CMD_DO_SET_SERVO`". Depending on the values of the command the application needs to set the right PWM value for the correct servo.

4.6.1. PX4 MAVLink Message Handling

The flight stack needed to be changed to make the application receive `MAV_CMD_DO_SET_SERVO` commands, so it accepts and handles the commands correctly. MAVLink messages are received, transmitted and handled inside the mavlink module. This module contains a class called "MavlinkReceiver", which is responsible for receiving and handling incoming MAVLink messages. Depending on the message type, the message is either published to the whole system, via the observer pattern, (GHJV95) and/or immediate actions are taken. Since the `MAV_CMD_DO_SET_SERVO` is encapsulated inside an `COMMAND_LONG` message it is interesting to see in detail how this type of message is handled.
Inside the MavlinkReciever class the function `handle_message_command_long_(mavlink_message_t*msg)` decodes the incoming `mavlink_message_t` messages into a `mavlink_command_long_t` struct. First it does some basic checks, if the packet is addressed correctly, by checking the target system and target component ids. After that it creates a `vehicle_command_s` struct and fills it with the data from the `mavlink_command_long_t`. The created vehicle_command finally gets published via the vehicle_command-channel. One subscriber to the vehicle_command-channel

is the commander-module. It is capable of handling some vehicle_commands, but for all unsupported vehicle_commands it will send a mavlink_message back to the GCS warning about the unsupported command. The commander-module also generates a warning tone on the UAV. Because of the onboard application, which now supports this command, this behavior needs to be changed. To achieve this, the `VEHICLE_CMD_DO_SET_SERVO` needs to be added to the list of commands with are ignored by the commander-module. This can be seen in the Listing 3.

```
        case VEHICLE_CMD_PAYLOAD_PREPARE_DEPLOY:
        case VEHICLE_CMD_PAYLOAD_CONTROL_DEPLOY:
    case VEHICLE_CMD_DO_SET_SERVO:
                /* ignore commands that handled in low prio loop
                 ↪  */
                break;

        default:
                /* Warn about unsupported commands, this makes
                 ↪  sense because only commands
                  * to this component ID (or all) are passed by
↪  mavlink. */
                answer_command(*cmd,
                 ↪  VEHICLE_CMD_RESULT_UNSUPPORTED);
                break;
        }
```

Listing 3: Part of the handle_command() function of the commander-module

4.6.2. Daemon

To support the `VEHICLE_CMD_DO_SET_SERVO` an onboard application for the PX4 has been developed. It consists of a daemon that pools the vehicle_command-channel.

If a `VEHICLE_CMD_DO_SET_SERVO` command has been posted, it will handle the command and will move the according servo. The Daemon is programmed with a global variable, which gets checked every time someone tries to start the daemon and is set during the start of the daemon. If the variable is already set, the daemon can not be started a second time. This is done with the singleton-pattern (GHJV95).

4.6.3. Handling *vehicle_command*

The job of the handle_command function is to extract the PWM values from a `vehicle_command_s` struct and save it to an instance variable. The function can be looked up in Listing 4. First the type of the vehicle_command is checked, while everything else than `VEHICLE_CMD_DO_SET_SERVO` will be ignored. The PWM value will be checked to prevent servo damages in case of too high values. If the values are secure to use they will be saved to a struct, which will later be read out by the main loop.

```
void
Skynet::handle_command(struct vehicle_command_s *cmd)
{
    switch(cmd->command){
        case VEHICLE_CMD_DO_SET_SERVO:{
            int servonumber = (int)cmd->param1;
            int pwmvalue = (int)cmd->param2;

            if (pwmvalue > PWM_HIGHEST_MAX)
            {
                //Dont accept to high PWM values, since it could
                ↪ damage the servo
                warnx("Not setting PWM: %u, for Servo: %u since
                ↪ it is to high", pwmvalue, servonumber);
            }else{
                //Save the PWM value inside the struct and the
                ↪ number of the servo
                _pwm_output.values[servonumber] = pwmvalue;
                _lastServoNumberRecieved = servonumber;
                warnx("Skynet recieved servonumber: %u with
                ↪ pwmvalue: %u", servonumber, pwmvalue);
            }
            break;
        }
        default:{
            break;
        }
    }
}
```

Listing 4: handle_command function from the Onboard Application

4.6.4. Main Loop

Inside the main loop of the daemon the vehicle_command-channel gets polled. The polling is done via the poll()-function with a timeout of one second. If this would have been done with read() the application would loop until something happens on the vehicle_command-channel. This would freeze the complete system which would lead to a crash of the UAV. With poll() the operating system lets the application sleep and wake it up as soon as something gets posted on the vehicle_command-channel. The timeout of one second makes sure that the application will wake up at least every second, even if nothing gets posted. This is required to make the application capable to react on other commands, like stop, which will shut down the application. After a command has been posted the vehicle_command gets copied from the channel into a local `vehicle_command_s` struct. After that the command gets parsed by the handle_command()-function (see: 4.6.3). At last the servos are moved. The ioctl()-function of the operating system will be fed with the command `PWM_SERVO_SET`, the number of the servo and the PWM value. Parts of the main loop can be looked up in Listing 5. Note: Only one servo will be moved with this approach, which is okay since every vehicle_command only contains information about how to move one and not multiple servos.

```
while(!_task_should_exit){
    int poll_ret = poll(fds, 1, 1000); // we wait 1 secound for
    ↪ new data
    if (poll_ret <= 0)
    {
        //no new data just move along
        continue;
    }else{
        // got new data
        if (fds[0].revents & POLLIN) {
            struct vehicle_command_s command;
            //copy vehicle_command for the channel into local
            ↪ struct
            orb_copy(ORB_ID(vehicle_command), _command_sub,
            ↪ &command);
            //hand vehicle_command over for parsing
            handle_command(&command);
            //set the new PWM-value to the servo
            ret = ioctl(_servo_fd,
            ↪ PWM_SERVO_SET(_lastServoNumberRecieved),
            ↪ _pwm_output.values[_lastServoNumberRecieved]);
            if (ret != OK)
            {
                err(1, "PWM_SERVO_SET");
            }else{
                warn("Skynet Servo: %u set with pwm value: %u"
                ↪ ,_lastServoNumberRecieved,
                ↪ _pwm_output.values[_lastServoNumberRecieved]);
            }
        }
    }
}
```

Listing 5: Main Loop from the PX4 Application

4.7. Skynet - GCS-Application

The Skynet - GCS-Application is capable of:

- Displaying the stereoscopic video feed.
- Reading the head movement data from the Oculus Rift.
- Translating the head movement to servo movement.
- Generating MAVLink messages to move the servos.
- Transmitting the MAVLink messages via UDP to MAVProxy.

Mac OS is used as platform since I have the most programming experience with Objective-C and the Mac OS ecosystem.

4.7.1. General Workflow of the Application

Because of the nature of the tasks the application has to accomplish, the individual parts are loosely coupled. To achieve lose coupling, the singleton pattern (GHJV95) is widely used. Some tasks will have to work together and others will work past each other. To display the video feed for the oculus rift, only the video converters are used and no information from the Oculus Rift is needed. For telling the UAV how to move the servos more resources have to be used. First the Oculus Rift needs to be polled for the current position, the position then gets converted into PWM values for the two servos, the values needs to be encapsulated into a MAVLink message and in the final step the message needs to be transmitted via UDP to MAVProxy.

4.7.2. Application Settings

For convenience some needed variables are saved to disk and are loaded on application start. This includes:

- Device-IDs for the two video capture devices

- Port-number on which the UDP-Server listens
- The servo-numbers for pan and tilt
- Booleans if the PWM values needs to be inverted
- Min and max PWM values for the servos
- MAVLink target sytem and component ids

This has been implemented with a singleton which will load the values from the hard drive the first time it gets initialized. Writing of the values back to the hard drive needs to be triggered manually.

4.7.3. Settings

The variables provided by application settings can be modified via the "Settings" menu. The settings are divided into three sub menus: General, MAVLink and Servo-Setup.

Converter Settings

With the converter settings (Figure 4.9) the user can select the capture devices for the left and the right camera. After the devices have been selected a preview is displayed.

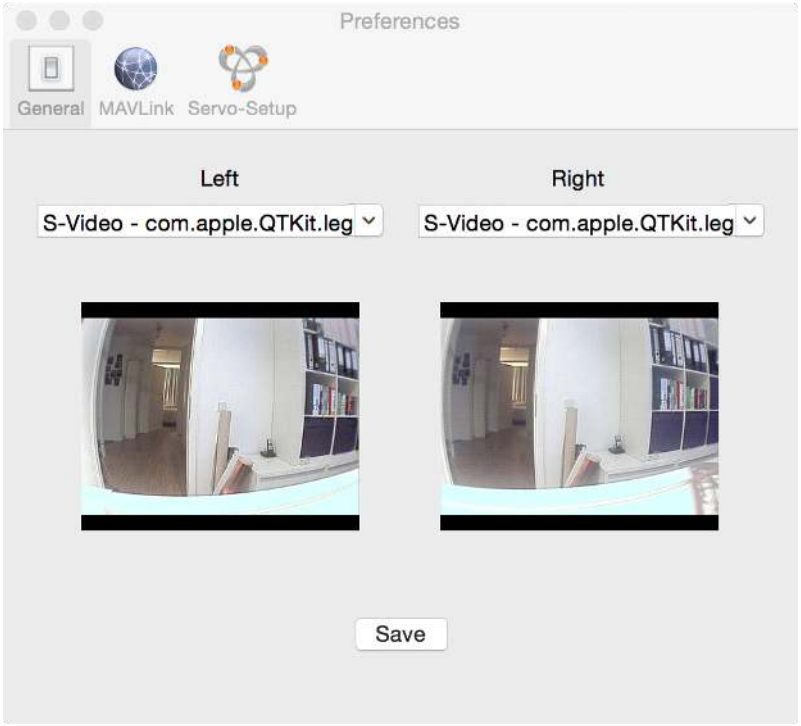

Figure 4.9.: Screenshot: Converter Setting

MAVLink Settings

The UDP-Port which is used to connect to MAVProxy can be set inside the MAVLink settings (Figure: 4.10). The MAVLink target system and component can be set as well.

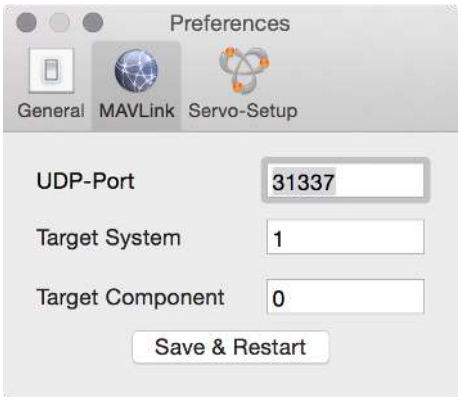

Figure 4.10.: Screenshot: MAVLink Setting

Servo-Setup

The Servo-Setup (Figure: 4.11) is used to configure the servos. For each movement axle, the maximum and minimum PWM value can be set in addition to the servo number and if the servo movement needs to be inverted. For easier setup the sliders will move the servos, where the left and right slider positions represent the extreme PWM values and the middle slider position the neutral PWM value.

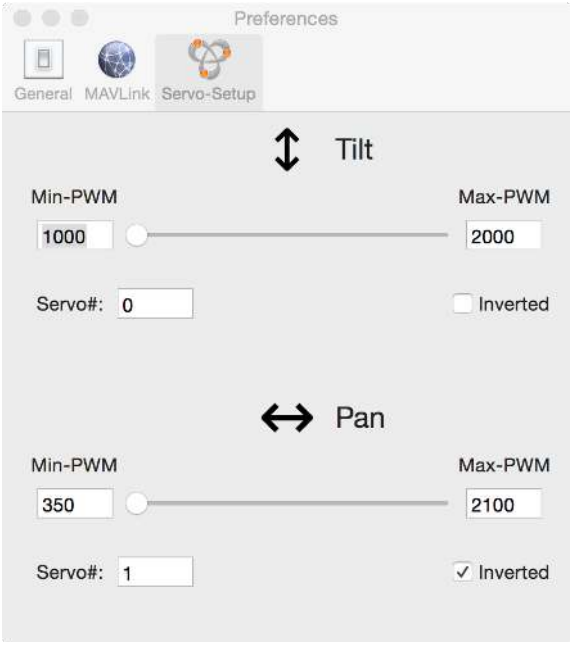

Figure 4.11.: Screenshot: Servo-Setup

4.7.4. Getting Data from Oculus

The HMDManager class is a singleton which will provide the positional data from the Oculus Rift. It gets initialized at application start. Three things happen during the initialization. The hardware of the Oculus Rift gets initialized first. Next, a timer will be created. It will fire the poolTrackerPosition() function every 10ms. At last, an event monitor will be created which will react when the "R" key on the keyboard is pressed. The Oculus Rift does not know where looking straight forward is. When the "R" key on the keyboard is pressed the Oculus Rift gets recentered on its current position.

_poolTrackerPosition()

This function uses the Oculus Rift Library to get orientation of the Oculus Rift on the x and y axes. The reported values are floats between -1 and 1, where 0 on the y

axis means that the Oculus Rift is looking straight forward. These values are then passed over to the CommCenter.

4.7.5. Calculating PWM Values

From the orientation delivered by the HMDManager the corresponding PWM values have to be calculated. The calculation is taking multiple variables into account: the degrees the servo can move, maximum and minimal PWM values and if the servo has been mounted in such a way that the resulting PWM values needs to be inverted or not. Those variables are retrieved from the Application Settings. One problem is that the Oculus Rift can be moved 360° on all axes, but the servos only support 180°. So every head movement over 90° to each side is capped to 90°.

To avoid flipping around between two PWM values, due to the sensitivity of the Oculus Rift Sensors, some precession is sacrificed. For reducing the amount of data that is transmitted to the UAV, the new PWM value is only transmitted if it differs from the previous PWM value for the servo.

4.7.6. Generating MAVLink Messages

Creation of MAVLink messages is done with the MavLinkMessageCreator which returns a ready to send NSData[18] object. Generation of the MAVLink message is done with the MAVLink C-Library[19]. A `mavlink_command_long_t` struct is generated and filled. Command is set to `MAV_CMD_DO_SET_SERVO`, param1 to the target servo, param2 to the PWM value, target component and system according to Application Settings. The struct is then encoded into a MAVLinkMessage. To create the NSData Object a buffer is created into which the MAVLinkMessage is written with the MAVLink C-Library. The NSData Object uses said buffer to create itself.

[18] `https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSData_Class/`

[19] `https://github.com/mavlink/c_library`

4.7.7. Sending MAVLink Messages

MAVLink messages are transmitted to MAVProxy. Communication between Skynet and MAVProxy runs over the network via UDP. CommCenter passes the generated MAVLink Message within an NSData Object to its UDPClient. MAVProxy has no option to create a port where it just listens for incoming MAVLink messages. It only allows to define a destination where messages are forwarded to. Since UDP is used no connection is established on which Skynet could send packages back to MAVProxy. But the UDP protocol defines that the source port of an incoming UDP packet can be used as a destination port for an answer (Pos80).

Server

The only task of the server is to accept packets from MAVProxy, just to extract the source port number of the packet and to save for the UDP client.

Client

The Client establishes a UDP connection to MAVProxy and sends the bytes from a NSData Object over the connection. On initialization of the client a c-socket is created. The created socket is setup to use UDP and the target port which was previously saved by the UDP server. Every time a packet is send, the created socket is used. Once the client gets deallocated the socket is closed. Should the port the UDP server saves change, the socket is closed and reopened.

4.7.8. Displaying the Video Feed

Inside the main view of Skynet the two video streams are displayed side by side in a window with the size of the native resolution of the Oculus Rift. Initialization and display of the video is done with the QuickTime Kit Framework (QT). For each USB video converter, a QTCaptureDevice [20] will be created with the deviceUID of

[20] `https://developer.apple.com/library/mac/documentation/QuickTime/Reference/QTCaptureDevice_Class/index.html`

the USB video converter. The ids are retrieved from the applications settings. Two QTCaptureSessions[21] are used to link the QTCaptureDevice to a QTCaptureView[22]. The two QTCaptureViews are placed inside the main view and display the video streams. Figure 4.12 shows a screenshot of the main view. The barrel distortion described in 2.2.2 does not need to be applied since the cameras used produce a barrel distorted image.

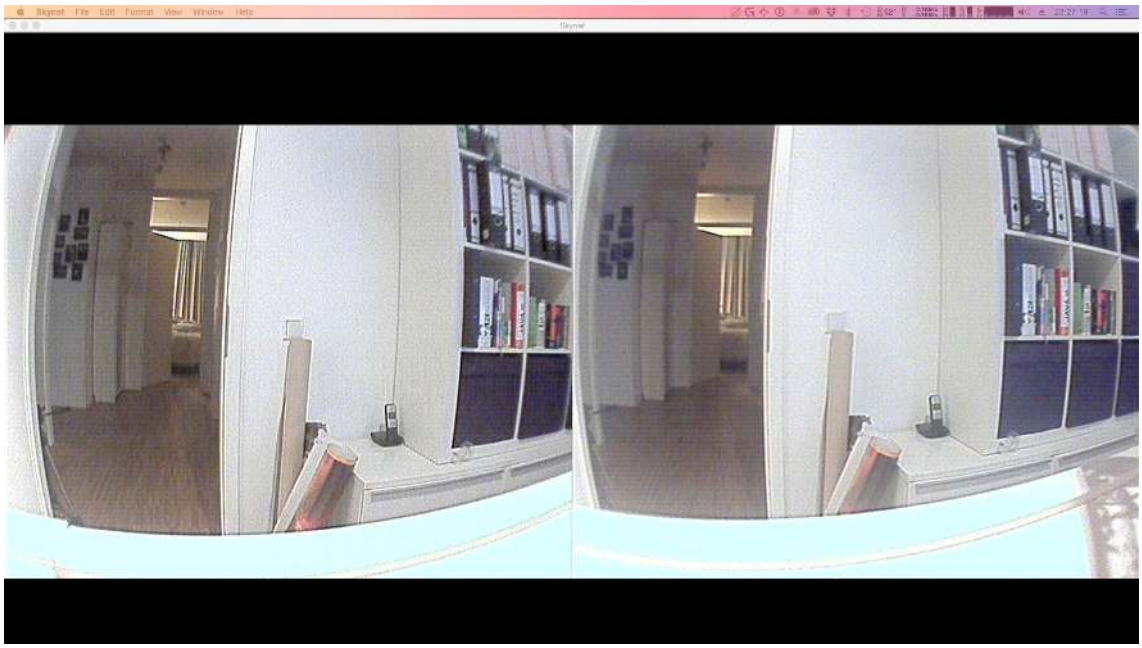

Figure 4.12.: Screenshot: Skynet

[21] https://developer.apple.com/library/mac/documentation/QuickTime/Reference/QTCaptureSession_Class/index.html

[22] https://developer.apple.com/library/mac/documentation/QuickTime/Reference/QTCaptureView_Class/index.html

5. Evaluation

It was the goal to build an autonomous flying drone that can survey a given area. The user should be able to easily look around, just like he is hanging on bottom of the drone. To improve the visibility of a missing person the user should be able to see stereoscopic.

For the autonomous hovering or flight along a predetermined path a hexacopter was build. It is powered by an autopilot that is capable of autonomous flight and hovering. To provide a stereoscopic video feed two cameras have been mounted to the bottom of the drone. The camera mount is constructed in such a way that the cameras can pan and tilt in 180°. For easy usage the user simply needs to put on a VR headset which will show him the stereoscopic live video feed. The movement of his head will be captured by the VR headset and interpreted by a self-developed software. Depending on the head movement, the software will generate matching control commands that are then send to the drone. On the drone an other self-developed program will process the control commands and put them into camera motion.

When the goals are compared with the outcome of the project, it can be considered a success.

In review two smaller blunders can be found. Since i had no experience with drones or R/C-Vehicles in general, the time needed for choosing the best components was to long and some components had to be switched out, for example the camera mount, during the development of the UAV. The other item would be the degrees of freedom the cameras can be moved. The tilt movement is fine, but for pan a greater degree of freedom, around 360°, would have been more desirable. But this might be fixed with a servo that is capable of moving 360° that would replace the current one.

6. Conclusion

In the introduction a brief introduction has been given how drones are slowly becoming part of our daily live. Then the reasons why VR seemed to have flopped in the early 90s and why it is now gaining more traction has been given. At the end of the introduction the motivation behind this thesis was explained. The second chapter gave an insight on what kind of UAVs exist, what the Oculus Rift is and how other projects already tried to combine these two technologies. In the design chapter the requirements for the system has been explained. After that, the required components and their task have been analyzed. For each component the part which has been chosen to be used is described in detail. The end of the chapter makes clear that two pieces of software are missing and illustrates what their tasks are. The fourth chapter provides an overview of the fully assembled system. The next part explains the construction of the drone in detail, followed by the ground station setup. The last parts of the chapter analyzes the developed program for the ground station and the onboard program for the PX4.

6.1. Further Prospects

To improve the stability of the produced video feed, a gimbal could be used to stabilize the cameras. This could be done by either by placing a hardware gimbal between the camera mount and the UAV or by extending the onboard application. The application would have to read the sensors of the UAV and determine its position in space and use this information to calculate an offset for the servos.

The cameras used are normal cameras. Either the cameras could be turned to infrared or night vision cameras to view during the night, or those cameras could be added to the camera mount and the transmitted video signal be switched between the different types.

To provide the user a greater degree in which the user can look around, not only the cameras should be moved, but also the whole UAV could turn in the desired direction.

A. Source Code

The source code can be found on the enclosed CD-Rom.

Bibliography

[AGS12] Angelo G Solimini, Domitilla Di Thiene Giuseppe La T. Alice Mannocci M. Alice Mannocci: A survey of visually induced symptoms and associated factors in spectators of three dimensional stereoscopic movies. (2012), 09

[dji] *DJI 2.4G Bluetooth Data-link Specifications*. http://www.dji.com/product/ipad-ground-station/spec, . – Last checked on 31.01.2015

[EH14] Erik Hals, Mats Krüger Svensson Mads Falmår W. Jacob Prescott: *Oculus FPV*. https://github.com/Matsemann/oculus-fpv/raw/master/abstract.pdf, 2014. – Last checked on 31.01.2015

[GHJV95] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns*. Boston, MA : Addison-Wesley, 1995. – ISBN 0201633612

[JN] Jack Nicas, Colum M.: *Who Builds the World's Most Popular Drones?* http://www.wsj.com/articles/who-builds-the-worlds-most-popular-drones-1415645659, . – Last checked on 31.01.2015

[MPB14] Marcin Piotr Biernacki, Łukasz D.: Mood and simulator sickness after truck simulator exposure. In: *International Journal of Occupational Medicine and Environmental Health* 27 (2014), 04

[MT14] Masumi Takada, Yasuyuki Matsuura Motohiko Sato Hiroki T. Yuta Fukui F. Yuta Fukui: Peripheral viewing during exposure to a 2D/3D video clip: effects on the human body. In: *Environmental Health and Preventive Medicine* (2014), 12

[Pos80] Postel, J.: *RFC 768 - User Datagram Protocol*. 08 1980

[rif] *Oculus Developer Guide*. http://static.oculus.com/sdk-downloads/documents/Oculus_Developer_Guide_0.4.4.pdf, . – Last checked on 18.02.2015

[RK10] Rainer Klinke, Rosemarie B.: *Physiologie: Lehrbuch.* 6. Edition 2010. Thieme, 2010. – ISBN 9783137960065

[SAE14] *JAUS/SDP Transport Specification.* http://standards.sae.org/as5669a/, 08 2014

[sam] *New Oculus Rift dev kit uses the front of a Galaxy Note 3 as its screen.* http://www.theverge.com/2014/7/31/5956589/new-oculus-dev-kit-uses-front-of-galaxy-note-3-for-display, . – Last checked on 18.02.2015

[Sch14] Schlechtriem, Gregor: *Ansteuerung von Modellbauservos (German Edition).* http://amazon.com/o/ASIN/B00ERC3INI/. Version: 2, 7 2014

[SLH11] Schmidt, Robert F. (Hrsg.) ; Lang, Florian (Hrsg.) ; Heckmann, Manfred (Hrsg.): *Physiologie des Menschen: mit Pathophysiologie (Springer-Lehrbuch).* 31. Edition 2011. Springer, 2011. – ISBN 9783642016509

[TK07] Tohru Kiryu, Richard HY S.: Sensation of presence and cybersickness in applications of virtual reality for advanced rehabilitation. In: *Journal of NeuroEngineering and Rehabilitation* (2007), 09

List of Figures

List of Source Codes

Abbreviations

DC ..... Direct Current

ESC ..... Electronic Speed Control

FMU ..... Flight Management Unit

GCS ..... Ground Control Station

GPS ..... Global Positioning System

HDMI .... High-Definition Multimedia Interface

Hz ..... Hertz

LED ..... Light-Emitting Diode

LiPo ..... Lithium Polymer

MAVLink Micro Air Vehicle Link

ms ..... Milliseconds

OLED .... Organic Light-Emitting Diode

PC ..... Personal Computer

PCB ..... Printed circuit board

PWM .... Pulse-Width Modulation

R/C ..... Radio control

RTOS .... Real-Time Operating System

SAR ..... Search and rescue

UAV ..... Unmanned Aerial Vehicle

UDP ..... User Datagram Protocol

USB ..... Universal Serial Bus

VR ..... Virtual Reality