X-Copter Team

Hochschule Ulm

FINAL REPORT

X-Copter

Table of contents

[2 Project Description 1](#_Toc440026086)

[3 Project members 2](#_Toc440026087)

[4 Analysis of customer needs 3](#_Toc440026088)

[4.1 Old Definition of Requirements: 3](#_Toc440026089)

[5 Active Customer Needs: //TODO haben sich die anforderungen geändert!? sind neue Dazu gekommen!? 4](#_Toc440026090)

[6 Project Management 5](#_Toc440026091)

[6.1 Scrum 5](#_Toc440026092)

[6.2 Roles in Scrum 5](#_Toc440026093)

[6.2.1 Product Owner 5](#_Toc440026094)

[6.2.2 Development Team 5](#_Toc440026095)

[6.2.3 Scrum Master 5](#_Toc440026096)

[6.3 Sprint 1 5](#_Toc440026097)

[6.4 Sprint 2 //TODO after sprint 2!? 7](#_Toc440026098)

[6.5 Milestone 3 //TODO 8](#_Toc440026099)

[7 Interim Evaluation by the team 8](#_Toc440026100)

[8 Project Documentation 9](#_Toc440026101)

[8.1 System Architecture //TODO 9](#_Toc440026102)

[8.2 First Test flight // TODO zu viele kleine abschnitte…viel mehr beschreiben 9](#_Toc440026103)

[8.2.1 Preparation 9](#_Toc440026104)

[8.2.2 Attempt 1 9](#_Toc440026105)

[8.2.3 Attempt 2 9](#_Toc440026106)

[8.2.4 Attempt 3 10](#_Toc440026107)

[8.2.5 Attempt 4 10](#_Toc440026108)

[8.2.6 Conclusion 10](#_Toc440026109)

[8.3 Second Flight Test // TODO noch schlimmer 10](#_Toc440026110)

[8.3.1 Organisation 10](#_Toc440026111)

[8.3.2 Changes at the X-Copter since the last Flight test 10](#_Toc440026112)

[8.3.3 Attempt 1 10](#_Toc440026113)

[8.3.4 Conclusion 10](#_Toc440026114)

[9 Hardware 11](#_Toc440026115)

[9.1 Power supply 11](#_Toc440026116)

[9.1.1 Get the circuit plan 11](#_Toc440026117)

[9.1.2 Check the board 11](#_Toc440026118)

[9.1.3 Final Solution 11](#_Toc440026119)

[9.2 Choice of the RC – Controller and the receiver 12](#_Toc440026120)

[9.2.1 The requirements for the RC- Controller and the receiver are: 12](#_Toc440026121)

[9.2.2 A selection of companies that produce RC- Controller: 12](#_Toc440026122)

[9.2.3 Major properties of the RC- Controller: 12](#_Toc440026123)

[9.3 Commercial Flight control 13](#_Toc440026124)

[9.4 Kinect and USB Host Controller Documentation 13](#_Toc440026125)

[9.4.1 Initial Situation 13](#_Toc440026126)

[9.4.2 Requirements for the USB-Controller 13](#_Toc440026127)

[9.4.3 Common USB-Controller Packages 14](#_Toc440026128)

[9.4.4 Controllers that come into question 14](#_Toc440026129)

[9.4.5 Conclusion 16](#_Toc440026130)

[10 Linux System 17](#_Toc440026131)

[10.1 Build Root 17](#_Toc440026132)

[10.2 Wi-Fi Connectivity 17](#_Toc440026133)

[11 Flight Controller 19](#_Toc440026134)

[11.1 Domain model 19](#_Toc440026135)

[11.1.1 Layer 1 19](#_Toc440026136)

[11.1.2 Layer 2 19](#_Toc440026137)

[11.1.3 Layer 3 19](#_Toc440026138)

[11.1.4 Layer 4 19](#_Toc440026139)

[11.1.5 Layer 5 19](#_Toc440026140)

[11.1.6 Layer 6 19](#_Toc440026141)

[11.1.7 Layer 7 19](#_Toc440026142)

[11.1.8 Layer 8 19](#_Toc440026143)

[11.2 Graupner HoTT-SUMD-Signal 21](#_Toc440026144)

[11.2.1 Why SUMD 21](#_Toc440026145)

[11.2.2 Definition 21](#_Toc440026146)

[11.2.3 Structure of a HoTT- SUMD frame 21](#_Toc440026147)

[11.2.4 SUMD\_Header section description 21](#_Toc440026148)

[11.2.5 SUMD Data section description 21](#_Toc440026149)

[11.2.6 SUMD\_CRC section description 21](#_Toc440026150)

[11.2.7 Channel data interpretation 21](#_Toc440026151)

[11.2.8 Implementation of the SUMD Parsing 22](#_Toc440026152)

[11.2.9 Saving raw SUMD-Frame Bytes from the UART 22](#_Toc440026153)

[11.2.10 Interpreting the received SUMD-Frame 22](#_Toc440026154)

[11.3 UART Driver 22](#_Toc440026155)

[11.3.1 SUMD-Frame-high 22](#_Toc440026156)

[11.3.2 SUMD-Frame-low 23](#_Toc440026157)

[11.4 PID Regulator 23](#_Toc440026158)

[11.4.1 The Proportional Part 23](#_Toc440026159)

[11.4.2 The Integral Part 23](#_Toc440026160)

[11.4.3 The Differential Part 24](#_Toc440026161)

[11.5 Use of the sensor board of Sparkfun 24](#_Toc440026162)

[11.5.1 Calibration of accelerometer 24](#_Toc440026163)

[11.5.2 Calibration of Gyro meter 25](#_Toc440026164)

[12 Lesson Learned //TODO 100 mal zu wenig 25](#_Toc440026165)

[12.1 Scrum 25](#_Toc440026166)

[12.2 Long term calendar management 25](#_Toc440026167)

[12.3 Small tasks take more time than thought 26](#_Toc440026168)

# Project Description

“In today’s modern high-tech world there are more and more fields of application for so called multicopters. These are aircrafts that are driven by multiple rotors, are relatively easy to control and are able to hover on one spot which even allows them to navigate through narrow and inaccessible areas. Typical applications of multicopters can be found in several areas like photography, emergency management and even parcel delivery. Thereby

The aircraft has not to be necessarily controlled by a human with help of a remote control. Instead there are also approaches in which the aircraft navigates through the area autonomously.

Outdoor the aircraft navigation is relatively easy because the aircraft can locate itself easily via GPS. The indoor navigation becomes far more complicated because GPS doesn’t work indoor since the system has to have visual contact to at least 3 satellites. Thus other navigation techniques like visual navigation over cameras and depth of field sensors have to be used which are intensively researched at the moment.

These alternative navigation techniques require very high processing performance. A common approach is to split the necessary computations on multiple independent processor cores. This leads to the problem that the different cores should access separate memories in order to avoid access contentions. These conflicts would reduce the performance and complicate the predictability within a real-time system. Therefore the memories of the different processor cores should be separated or alternatively each processor has to have a well dimensioned cache.

In order to allow research in this field, a system should be built up preferably modular so that various sensors and actors can be installed for testing. A system for such research purposes should be built up in this project. Detailed information regarding the project goal can be found in the next section.” [1]

The previous team developed a model and some software for the X-Copter. Our goal is to continue to develop more Software and test the flying ability of the X-Copter. Another goal is to get the data while the X-Copter is in the air. This will be achieved through an external monitoring on the ground.

# Project members

* **Jan Goller**
  + Power supply
  + Linux
  + WIFI
  + Buildroot
* **Thomas Weber**
  + Flight Controller
  + RC Transmitter
  + RC Receiver
  + Sensor Calibration
* **Alexander Ott**
  + Flight Controller
  + Construction
  + Sensor Data Managment
* **Florian Schneider**
  + Flight Controller
  + Power supply
  + PID and Filter
* **Daniel Maurus** 
  + Linux
  + WIFI and WPA
  + MCAPI
  + Accumulator
* **Stephan Gabor** 
  + Linux
  + USB – Controller
  + Accumulator
  + MCAPI
* **Jochen Hoeft**
  + Pilot
  + Flight Controller
  + RC Receiver
* **Lukas Öfner**
  + Scrum Master
  + Communication customer/teams
  + MCAPI and MAVLINK

# Analysis of customer needs

## Old Definition of Requirements:

"The system has to carry a payload of minimum 1 kg. Therefore normally 6 to 8 rotors are needed and it has to be evaluated which number of rotors fits our requirement best by measuring the lifting capacity of selected motors and rotors.

Another requirement of the customer is that the model should fit through a standard door. Because of this the model has to be constructed with a maximum width of 85cm to have enough clearance.

The system also has to reach a flight time of 10 to 20 minutes. For multi-copters normally Lithium Polymer accumulators with 1 to 10 cells and 500 to 20000 mAh are used and it has to be measured how much power is consumed by the system, especially the motors. To reach these requirements the weight of the model should be as lightweight as possible. All components have to be checked regarding to their weight and the use of different materials such as carbon fibre should be evaluated.

The Customer also wants to have a modular design of the whole system.

Therefore a physical model has to be designed in order to have enough space for additional modules such as new sensors. It also has to be possible to change the weight distribution to keep the model balanced and also the electronics need to have enough standard interfaces to add new hardware components.

Another requirement of the customer is that the different software components don’t interfere with each other. To fulfil this, the system should consist of multiple processors that have separated memory and interact with each other over bridges. The customer also wants to have the possibility to extend the existing multiprocessor system with more powerful hardware over a widely spread communication protocol. To meet this requirement an Ethernet interface should be realized and the system should support to give the new hardware access to the required sensors. To meet security requirements manual interaction has to be possible at all times. Therefore the system has to have a receiver for a remote control and has to meet hard real-time requirements.

The system also should be able to fly stable and to give the other processors the possibility to control the flight of the system. Therefore a flight control unit has to be designed that has an interface with which other processors can interact." (Team-Bumblebee, 2015)

# Active Customer Needs: //TODO haben sich die anforderungen geändert!? sind neue Dazu gekommen!?

Since this project was transferred to us. The Basic hardware functionality was completed by our predecessors. As well as an interprocessor communication system.

Thus the customer needs can be extended with the following requirements.

The first demand was to ensure that the basic construction of the Multi copter is working properly. In order to receive fast test results, a commercial flight controller has to be integrated. Another requirement is to provide a power supply that is capable of delivering sufficient power for all components. One of the main requirements was previously described as "The system should be able to fly stable and to give the other processors the possibility to control the flight of the system".

The first version of the flight controller shall be offering the bare minimum functions to fly the Quadcopter model. However it has to be designed to offer improve its stabilization capability by more sensor data (e.g. magnetometer, barometer). Furthermore it should provide a connection to a ground monitoring station, using a communication protocol which was originally developed for a commercial Flight controller called "Pixhawk".

The connection to the monitoring station should be established via Wi-Fi.

Another task for the HPS will control the 3D cameras.

# Project Management

## Scrum

Scrum is an iterative and incremental agile software development methodology for managing product development. We had a lecture of Project Management by Dr. Balsen. He owns a small developing company and he has used Scrum for years. Dr. Balsen showed us all the positive and negative aspects of Scrum and was teaching us how to handle Scrum. A key principle of scrum is its agile switch of customer changes during a project. And another main reason for Scrum is the step by step developing.

## Roles in Scrum

### Product Owner

The product owner is the voice of the customer. He writes customer-centric items typically, the user stories, ranks them and prioritizes them. In our project the product owner is same as the customer.

### Development Team

The development team is self-organizing in Scrum. A team is made up of 3-9 members. The team is responsible for the progress of the project. Each team had their own tasks.   
In each task the actual work is described by the product owner and scrum master. If the development team finishes some tasks the continuing with the next open task.

### Scrum Master

The scrum master is coaching the team with the scrum principles. He is responsible to remove impediments of the development team. The scrum master facilitation team events like the daily scrum or other meetings. He acts as a buffer between the team and the customer.

## Sprint 1

The first Sprint started at the 13. April and ended on the 11. June. We finished a lot of tasks and reached almost all our goals completely. You can see it in the Burndown Chart below in Figure 2 Burndown-Diagram Sprint 1. The last part where the curve doesn’t fit to the nominal value was the issue with the USB-Controller. The Problems are documented in the Impediment Backlog. All Sprint tasks are documented exactly below.

1. Select the remote:

* costumer pitch about the remote
* comparison of remotes
* price inquiry
* ordering the remote

1. Stable battery

* getting the [circuit](http://www.dict.cc/englisch-deutsch/circuit.html) [diagram](http://www.dict.cc/englisch-deutsch/diagram.html)
* checking the existing board
* building the circuit on a prototype board

1. Charging [the](http://www.dict.cc/englisch-deutsch/the.html) [battery](http://www.dict.cc/englisch-deutsch/battery.html)

* programming the charger
* extern power supply

1. USB-Controller

* comparison of USB-Controllers
* requirements
* selecting a new USB-Controller

1. Commercial flight controller

* get the Software and install it
* feature list
* configure the cruise [control](http://www.dict.cc/englisch-deutsch/control.html)
* connect the cruise controll to the rotors
* connect the cruise controll to the flight controller
* connect the flight controller to the X-Copter
* get a connection to the remote
* configurate the flight controller
* configurate the remote

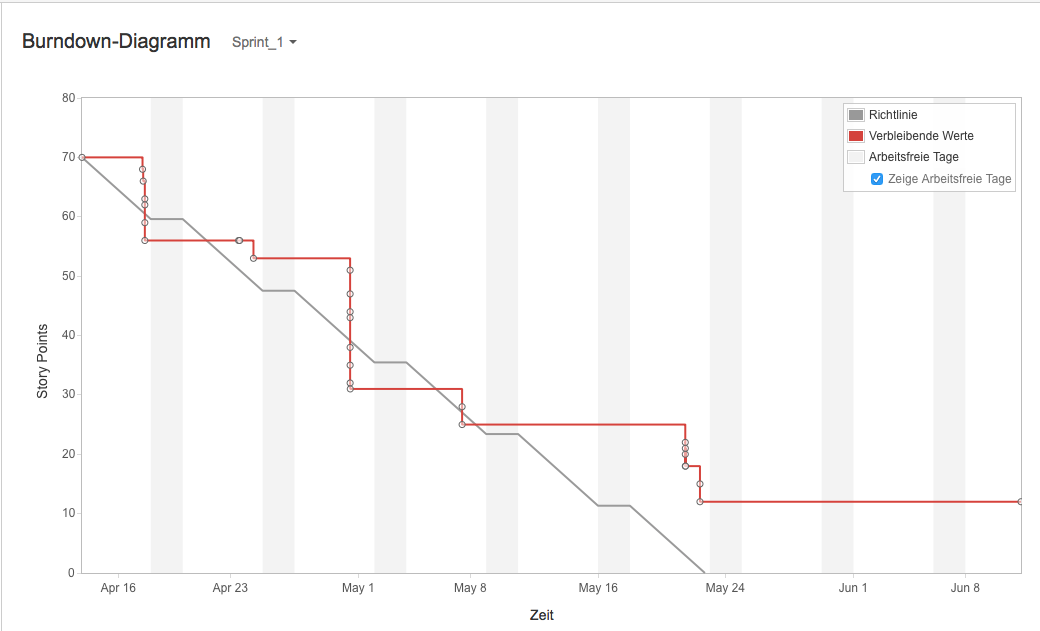


Figure Burndown-Diagram Sprint 1

## Sprint 2 //TODO after sprint 2!?

1. Commercial flight controller

* Back up the configuration of the controller
* Get a date for the first flight
* Organize a bus for the test flight
* Switch for the RC-Controller
* Build a frame for the landing
* Fuses for the X-Copter

1. Simple flight controller

* Get information about the drivers
* Reveal the components of a flight controller
* Get information about PID regulator
* Test the drivers

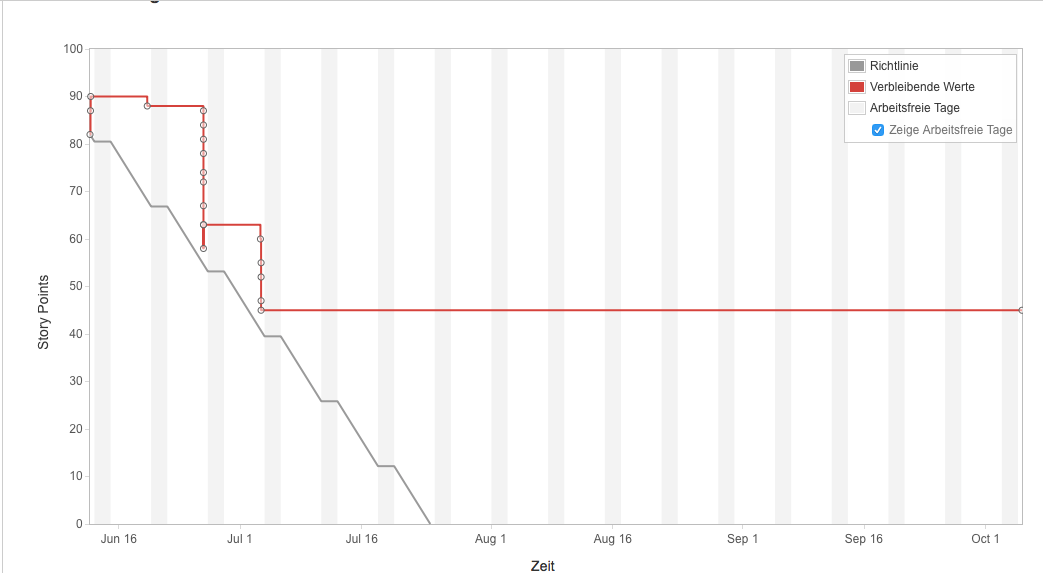
1. USB-Controller

* Search for a new USB-Controller

1. Day of informatics

* Make a presentation about the X-Copter

Make a poster for the day of informatics

 Burndown-Chart Sprint

## Milestone 3 //TODO

# Interim Evaluation by the team

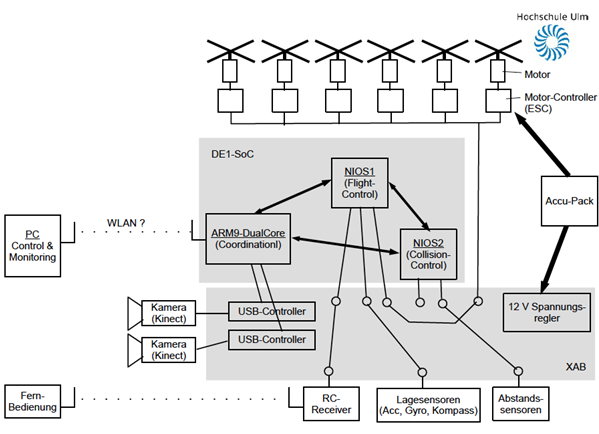
In the last steps we reached some important milestones like the first successful flight or the successful transmission from the RC Transmitter to the NiosII-processor. On the way to these goals, our team worked very well together and everybody solved his own tasks very carefully. So the team was able to finish a lot of small tasks in a short time. However, at some tasks, we have also lost a lot of time. For example, the integration of the additional USB-controller, the error analysis of the existing power supply and the installation of the Wi-Fi-Driver on the embedded Linux. Another problem was our development methodology - Scrum was a loss of time. We decided to focus on the software development and not on scrum anymore. Thus we should be able to get more work done, in a shorter period of Time in order to reach our goal

# Project Documentation

## System Architecture //TODO

The system architecture is documented in the final report of the team Bumblebee. (Team-Bumblebee, 2015) This is a quick overview for our project. The DE1-SoC board is the main part of the X-Copter. The board contain a ARM9-DualCore CPU and two NIOS1 CPUs. The extension board is for all Sensors and the power supply. The cameras are connected to the extension board.

Our RC-Receiver is also connected to the extension board.



## First Test flight // TODO zu viele kleine abschnitte…viel mehr beschreiben

### Preparation

At first all Components got attached to the Model.In addition to the aircraft, a basic equipment of tools and spare parts was taken to the Model flying site.

### Attempt 1

The first attempt failed. The Motors had the wrong direction of rotation. So there was now upwards boost but a downwards boost.   
Error Analysis: Falsely it was assumed that the rotors were mounted incorrectly. So we switched the Rotors which led to upward boost from every rotor.

### Attempt 2

The wrong error analysis led to another fail attempt. From the perspective of the flight controller now each rotor turned the wrong direction. Now the yaw correction worked the wrong direction and self-reinforced the rotation of the X-Copter. The vehicle came immediately out of control.

Error Analysis: The spinning direction of all motors were checked and we noticed the wrong spinning direction. The reason for this was that the flight controller has been mounted rotated by 90∞. The flight controller was turned by 90∞ and remounted. In addition, all rotors were switched back in the correct position.

### Attempt 3

This attempt started well but the X-Copter lost upward boost at one side and crashed. Error Analysis: It turned out that one rotor loosened. All rotor screws were tightened.

### Attempt 4

The Take-off went well. However, one Motor flew away. Error Analysis: The rubber vibration damper that were used to mount the motors seemed to be not strong enough. The tools and spare parts we had were not enough to mount the motors without the vibration damper.

### Conclusion

The X-Copter vehicle should be able to fly. The rubber vibration damper is not recommended. Another Test is necessary.

## Second Flight Test // TODO noch schlimmer

### Organisation

The Organisation was similar to the first flight test. For security reasons we went to a Model flying site. Again the control was taken over by Mr. Hoeft because his Model flying insurance.

### Changes at the X-Copter since the last Flight test

To mount the motors without the rubber vibration damper the mount points had to be modified slightly. The motor direction of rotation has been checked.

### Attempt 1

Even though the conditions were bad (it was very windy this day) the X-Copter flew very well. The Test was a success.

### Conclusion

In quadcopter form the X-Copter does only use about 50% of its power, so there is a lot of capacity for payload left.

Now it’s proven that the Model is able to fly and we can start to work seriously with our own flight controller.

# Hardware

## Power supply

Because the power supply that was available from the older BumbleBee-Project, isn’t working, the need for a new, functioning power supply came up.

### Get the circuit plan

To get a circuit plan, the files and data from the BumbleBee-Project, were taken.   
These files contained an EAGLE ® formatted plan of the circuit.

### Checking the circuit for errors

The old power supply circuit was generated by a web app by Texas Instruments ® called WEBENCH® System Power Architect. A new circuit plan was generated by this tool with the following input parameters:

* V\_in\_max : 25 V
* V\_in\_min : 13 V
* V\_out : 12 V
* I\_out : 5 A

These parameters are given by the maximum and minimum output of the accumulators (V\_in) that are used on the XCopter and the SoC-Board restrictions, which are 12V input voltage and 3.5A output current. For provision 1.5A was added to the I\_out parameter.

After comparing the new circuit plan with the old plan, the last project used it was figured out, that they are the same so there cannot be the problem.

The next step was to compare all the components and the voltage control IC. There it was discovered that the last group, which designed and populated two circuit boards, used different ICs on each of them. So at least one of the boards cannot work.

Big error sources are the SMD parts. The problem with them is that we cannot easily test them. Therefore all the parts were bought in DIP norm and the circuit was built on a plugboard. The voltage control IC wasn’t available in DIP norm so it was mounted on an adapter to use it on the plug board.

Because all this cost a lot of time and our missing experience and tools in this subject, it was decided to buy a commercial power supply from an online shop.

### Final Solution

The final solution is a DC to DC voltage regulator which fulfills all the necessary parameter. It’s possible to adjust the Output voltage with a screwdriver. The regulator has the following features:

* V\_in: 5v - 30V
* V\_out: 5V - 12V
* I\_out: max 6A



## Choice of the RC – Controller and the receiver

### The requirements for the RC- Controller and the receiver are:

* Both have to use the sum signal
* Both have to provide 4 channels minimum, better up to 6 – 8 channels
* The costs have to be less than 350€ for both of them
* The RC- Controller have to be configurable easily

### A selection of companies that produce RC- Controller:

1. Graupner
2. Futaba
3. Spektrum
4. Modelcraft

It was decided to work with a Graupner RC- Controller!

**Reasons for Graupner:**

* Graupner is an innovative and leading company in RC- modelling
* Graupner ensures a high quality standart
* Graupner provides lots of datasheets for each product
* Graupner has a big RC- community

Out of the range of Graupner products the “ Graupner MX 16 “ was selected:

### Major properties of the RC- Controller:

* 8 channel
* HoTT technology (sum signal, transmit up to 16 channels)
* Bidirectional communication between transmitter and receiver
* Free configurable switches
* Signal range 4 km
* Very fast rebinding

Graupner MX16 <http://www.live-hobby.de/out/pictures/master/product/4/33116.jpg>

## Commercial Flight control

In order to test the construction of the XCopter for the first time, without wasting too much time on developing an own flight control, a commercial flight control was installed on the XCopter. In this case the DJI NAZA V2 was used.

It is a fully developed flight control unit, which was developed to be easily installed in any multi copter system. It comes with an integrated 3-axis gyro sensor and acceleration sensor as well as an external GPS unit. The only items which need to be connected to it are all Electronic speed controllers (ESCs) and a RC-receiver, the gimbal (DJI camera) part is not necessary.

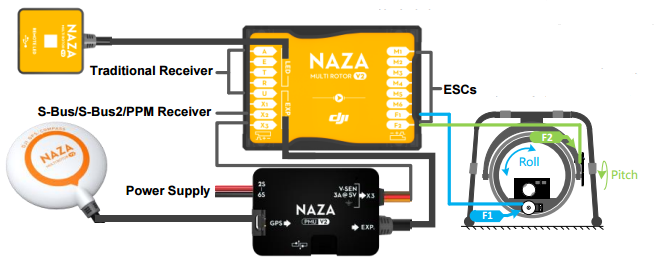


Figure 3 NAZ V2

Overview of DJI NAZA V2

http://download.dji-innovations.com/downloads/nazam-v2/en/NAZA-M-V2\_Quick\_Start\_Guide\_en.pdf

## Kinect and USB Host Controller Documentation

### Initial Situation

The customer wants two Kinects to be put on the X-Copter. They are to provide 3D-Image data to map the surrounding locale of the X-Copter. To be able to communicate with two Kinects, two USB Host Controllers are needed. This is because one Kinect needs at least ~21 MB/s data transfer rate for 3D-Images at 640×480 pixels with 30 frames per second, which is too much for one controller to handle. 21 MB/s are divided into ~12MB/s for depth camera and 9 MB/s for colour camera. For proper 3D-Image data colour- and depth camera have to work at the same time and cannot be separated, which strikes out the option to save bandwidth with using only one camera at the time.

Kinect cameras will be connected via USB 2.0 plug to the USB-Controllers. For the Controllers to be able to communicate with the DE1-SOC system, an interface has to be implemented into the existing SOPC for communication between the devices. Real time 3D-Data processing will be the task of another external system with an Intel processor. Our customer stated that on a similar side project of him even an Intel I7 quad core processor is struggling with processing the data. For further information about hardware requirements of Kinect-Systems refer to (msdn.microsoft.com).

### Requirements for the USB-Controller

There are certain cut in stone requirements for the USB-Controller to work with Kinect and to fit in the design of our system:

* Must be available on the market
* Must not exceed the quantity of pins our system is able to offer
* Drivers for Linux have to be available
* Chip has to have outgoing pins to be solder able
* Full High-Speed data transfer rate of 480 MBit/s
* (Should be ULPI compatible if present Waveshare 3300-Transceivers are meant to be used)

### Common USB-Controller Packages

There are three different common USB-Controller Packages that are solder able with the equipment available: QFN (Quad Flat No-leads package), LQFP (Low Profile Quad Flat Package) and TQFP (Thin Quad Flat Package). Information, advantages and disadvantages of these packages can be reviewed at (Seifert). QFN is harder to solder which is why QFP style packages are the preferred choice.

### Controllers that come into question

Investigation about USB-Controllers lead to a list of four different controllers that will be evaluated further in this document. The first controller is one chosen from Frank Seifert for his Bachelor's Thesis: “Conception and realization of a control computer platform for a quadcopter flying model”[6]. He compared three different solutions for USB-Controller implementation into his system. His selection included the ISP1362BD, its successor the ISP1761BE and a softcore FPGA solution. Implementing the USB-Controller directly into the FPGA fell out of the question because of the high price for an USB-Controller IP-Core (prices circle around 5000€). Open Source IP Cores for USB Host Controllers are few, have a low set of features and are badly documented, which makes them less than optimal for this project. Frank Seifert also crossed out the ISP1761BE because of a higher pin count and no Linux drivers available at the time of writing his Bachelor's Thesis. His research let him to believe that the ISP1362BD would be best suited for his endeavors.

Further research from our side showed that Linux Drivers are available for the ISP1761BE nowadays, which would make it a suitable choice for the project. Further investigation showed that the successor to the ISP1761BE, the SAF1761BE from NXP Semiconductors, is also available to purchase and supported with Linux drivers. The fourth and last USB Controller mentioned here is the FT313H(L/P) from Future Technology Devices International Ltd.

Cypress is another company that is also offering a wide array of USB solutions, sadly they don't have USB 2.0 Host Controllers in their repertoire.

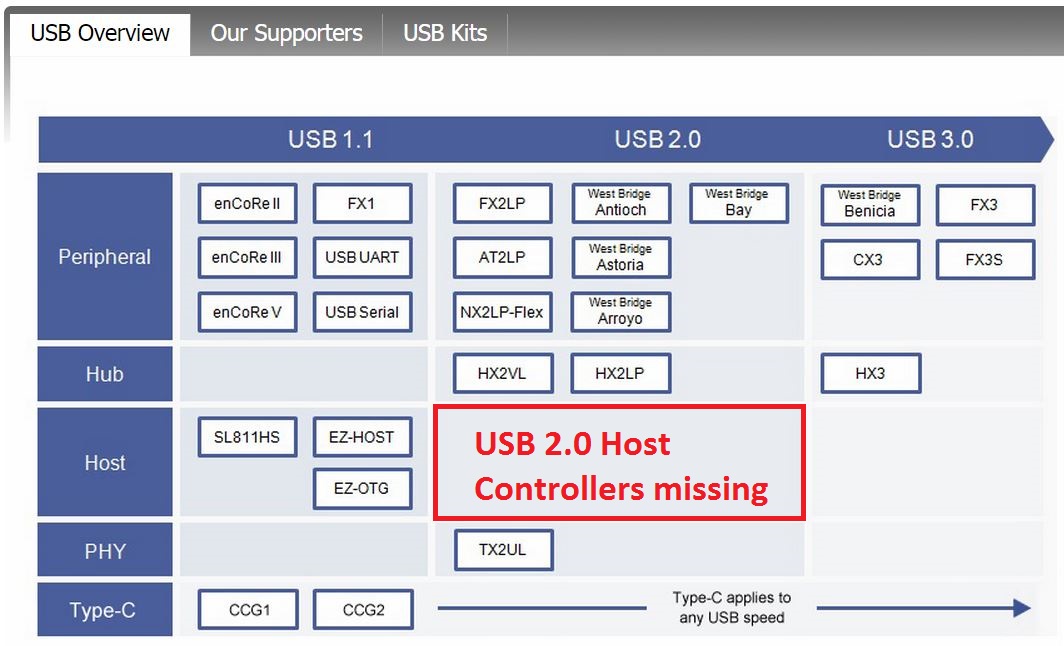


Illustration 1: Cypress USB offerings (http://www.cypress.com/fckImages/myresources/USBControllers\_Overviewimg(1).jpg)

Chips with packages that are not solder able with the equipment at our disposal will not go into the equation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **ISP1362BD** | **ISP1761BE** | **SAF1761BE** [3] | **FT313H(L/P)** [4] |
| **Date of production** | Rev. 04  12.2004 | Rev. 01  01.2005 | Rev.02  06.2012 | Ver.1.2  2013 |
| **Package** | LQFP64 | LQFP128 | LQFP128 | 64 LQFP  64 TQFP |
| **Driver for Linux** | yes | yes | yes | yes |
| **Transfer rate** | 96 Mbit/s | 480 Mbit/s | 480 Mbit/s | 480 Mbit/s |
| **RAM Memory** | unkn. | unkn. | unkn. | 48 KB |
| **ULPI compatible** | unkn. | unkn. | unkn. | unkn. |
| **Quantity of I/O Pins** | 27 Pins | 16Bit: 41 Pins  32Bit: 57 Pins | 16Bit: 41Pins  32Bit: 57 Pins | ? |
| **Info** | Discontinued | Discontinued | Available | Available |
| **Pros** | + Frank Seifert implemented this chip in his bachelor project | + speed  + similar to ISP1362BD | + speed  + similar to ISP1761BE | + speed  + 64 PIN package  + UMFT313EV Development Module available |
| **Cons** | - speed (too low for Kinect)  - not available | - not available | - used mainly in automotive systems  - no evaluation board | Unkn. As of time of writing |

USB Controller Comparison

### Conclusion

The ISP1362BD is not suitable as an USB Controller for the use with Microsoft Kinects because of a transfer rate of only 96 Mbit/s which is Full Speed USB 2.0. Kinects need at least High Speed USB 2.0 with 480 Mbit/s. Furthermore the controller is not supported anymore and it is almost impossible to obtain those controllers on today's market.   
The ISP1761BE does not make the cut either. Although it supports High Speed USB 2.0 and has Linux drivers, it has a larger footprint with its LQFP128 package and is also discontinued. Its successor the SAF1761 which is similar in features is mainly used in automotive systems which means that it is not available in the common consumer market .  
Which leads to the FT313H(L/P) which offers the best characteristics for our endeavours. It is still supported, offer Linux drivers. It has a relatively low footprint, is solder able with the tools at hand and comes in two packages: 64LQFP and64TQFP. It supports High Speed USB 2.0 transfer rates and can also be ordered with a development module.

# Linux System

## Build Root

Buildroot is an open source project that makes it possible to create an individual Linux system. It’s divided in different parts. Buildroot can automatically build the required cross-compilation toolchain, create a root file system, compile a Linux kernel image and generate a boot loader.  
Busybox and uClibc are the main parts of Buildroot. uClibc is a standard C-Library for embedded Linux systems. Busybox is a program which includes all necessary Unix services in one compressed package. It’s perfect to use for embedded systems with only limited resources.   
The HPS (Hard Processor System) has a first stage bootloader on the internal ROM. It scans the partition with id = a2 for the next stage bootloader. This second stage bootloader is limited in its size by 64kByte and starts the U-Boot-loader. Its task is to boot up the Linux system. Usually these boot loaders has to be configured once and not at any time when something has been changed in the Linux image.   
The main parts of the Linux system was built by Mr. Strahnen. The system runs on kernel 3.10.37, because of the long term support. It’s not possible do compile code on the ARM CPU because of the missing toolchain. Therefore it’s necessary to cross-compile the code on the virtual machine. (REF HowTo ARM/LINUX) Components like the WIFI driver were added via Buildroot.  
[2] [3]

## Wi-Fi Connectivity

The X-Copter needs a wireless connection to transmit/receive data to/from ground station. The data consists of position, status, speed and further information about the current air situation. The basic configuration is a PC/Laptop and a Wi-Fi dongle (“Edimax” with a RTl8188 chip) plugged in the X-Copter DE1-SoC board (DE1). Both are connected with an access point (AP).

First step to solve this challenge was to establish a connection between the DE1 and the AP. The Wi-Fi dongle doesn’t work out of the box. To get the dongle working there were two possibilities. First one was to compile a Linux driver and the second was to edit the operating system.

The driver can be downloaded from the manufacturer’s website. It was necessary to cross-compile the driver on the host x86 system for the target platform with an ARMv7 architecture. It’s a big underpinning to understand the makefile(s) and it is often not clear how to fix an error. After failing the task this way it was decided to edit the operating embedded system and include the drivers in “Buildroot”.

Buildroot was used to generate the embedded Linux, the bootloader and the root filesystem. At first the right device driver cannot be found in the basic settings. After activating some other components, additional devices were added to the Wi-Fi driver list. Now the driver file can be loaded successful but there was an error with a missing firmware file. Adding the right firmware in Buildroot solved this issue. As a precaution all Ralink drivers were activated.

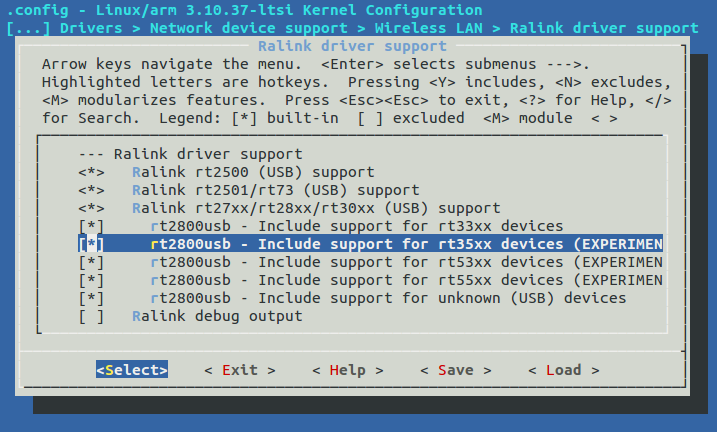


Figure 8 Wi-Fi Connectivity

Afterwards a connection failed because the DMA coherent pool is too small. Setting the variable of the coherent pool to 4 Megabytes in UBoot solved the problem.   
Now the basic settings are set and the DE1 can establish a wireless connection to the AP. The connection seems to be stable and it’s possible to send a ping signal to the DE1. At the moment the connection has to establish manually. In case of the time pressure it wasn’t possible to make a config file. In future the connection will establish automatically.

# Flight Controller

## Domain model

In the picture Figure 4 UML-Diagram Softwarethe layer orientated structure of our flight controller can be seen. From the top to the bottom, each layer uses the software form the layer below. This subjection is shown by the arrows pointing to the bottom. The very bottom layer is layer 1 and the very top layer is layer 8. The flight controller acts as an auto stabilizer when the UAV is in the air so that the user doesn’t have to steer against wind gusts and other external influences that disturb the flight.

### Layer 1

The very bottom layer is the layer that's closet to the hardware and coloured light red in Figure 4 UML-Diagram Software. In this layer there are the drivers for the general I/O communication on the board such as I²C and PWM.

### Layer 2

In the second layer which is coloured yellow, the drivers for the sensors and actors the X-Copter uses to communicate with the real world. Here four or six motors with rotors are used depending on the configuration of the X-Copter. In addition, there is an accelerometer, a gyroscope and a compass to detect the three dimensional lay and alignment. To share data between each other, they use the orange layer drivers. To receive the command signals from the remote control a RC driver is also implemented in this layer.

### Layer 3

To get useful data from the sensors it has to be filtered first to handle the disturbances. These filters are coloured dark green and are in layer three.

### Layer 4

In the fourth layer the RC and sensor interpreters are placed and coloured blue. The interpreters are used to get human readable and understandable data from the sensors.

### Layer 5

One layer above and coloured pink, there are the controllers for the sensors, motors and the remote control. Each controller manages the specific device. This includes accessing the driver to get or set the raw data from the drivers. This raw data is then, if it’s one of the sensors, filtered and interpreted. All this is done automatically to provide an easy access by the controllers to higher layers.

### Layer 6

The PIDs are located between the PID\_Controller (orange) and the other controllers (pink) in the sixth layer. Each motor has its own PID regulator that is controlled by the orange coloured PID\_Controller.

### Layer 7

The PID controller in layer seven collects all necessary data from the controllers in the pink layer five for the PID regulators. This includes the set point from the remote control and the actual value from the sensors. The value the PIDs return is then used by the motor controllers to set the needed speed to let the XCopter fly stable.

### Layer 8

The very top layer is the flight controller that manages the timing and general flow of the application in the main method.

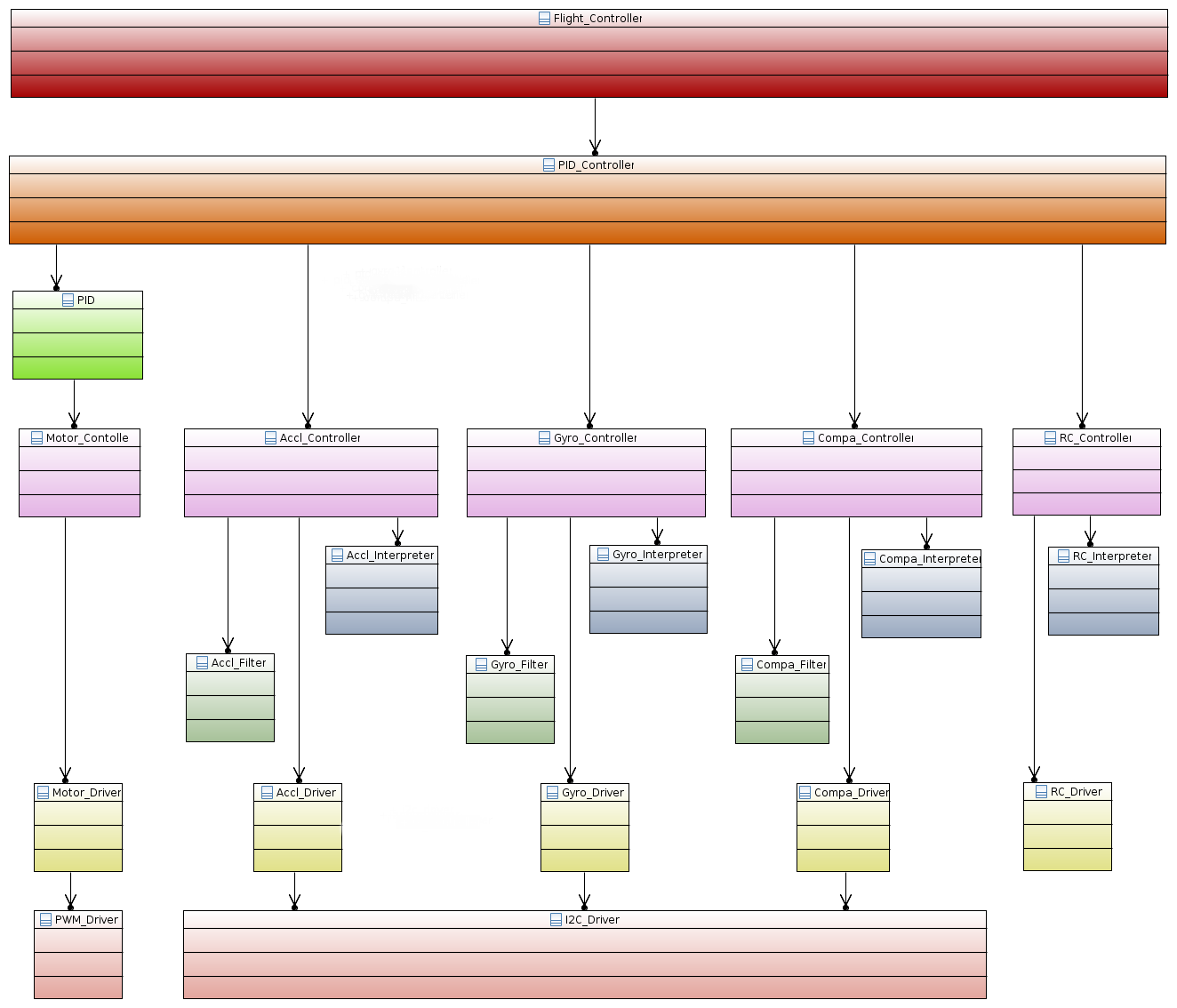


Figure UML-Diagram Software

## Graupner HoTT-SUMD-Signal

### Why SUMD

The “GR-16” receiver supports two different sum signals, “SUMO”- and “SUMD”- Signal. The SUMO- Signal is an analogue sum signal and is equal to a pulse position modulation (PPM) whereas the SUMD- Signal is a digital sum signal. So the big advantage of SUMD is, that it is easy decode able.

### Definition

HoTT SUMD is implemented with an 115200 bit/s serial data stream. The data stream is generated by HoTT receivers. The transmitter generates data frames at a data rate of 100Hz (10ms). Each data frame consists of a header followed by a data section representing the channel data and is concluded by a CRC checksum.

Each data frame is sent as a consistent data burst leaving minimal gaps less than 50µs between transmitted data bytes. The serial connection has to be setup whit the following:

* 115200 Bit/s baud rate
* 8 Databits
* no Paritybit
* 1 Stopbit

### Structure of a HoTT- SUMD frame

A single SUMD data frame comprises of three consecutive sections. SUMD\_Header, SUMD\_Data, SUMD\_CRC. The SUMD\_Data section contains the channel data in sequential order. The number of channels to be transmitted can be up to 32. Each channel data is represented by an unsigned 16 bit word.

### SUMD\_Header section description

|  |  |  |
| --- | --- | --- |
| ***Byte*** | ***Byte\_Name*** | ***Byte\_Value*** |
| Byte 0 | Vendor\_ID | 0xA8 |
| Byte 1 | Status | 0x01 or 0x81 |
| Byte 2 | Number of channels |  |

### SUMD Data section description

Byte n\*2+1 High Byte of channel n

Byte n\*2+2 Low Byte of channel n

### SUMD\_CRC section description

Byte (N\_Channels+1 )\*2+1 High Byte of CRC

Byte (N\_Channels+1 )\*2+2 Low Byte of CRC derived

### Channel data interpretation

Each channel data is represented by an unsigned 16 Bit Word. The data range is derived from the pulse length for standard servos.

|  |  |  |
| --- | --- | --- |
| Stick Position | Channel Data | Remark |
| Extended low position (-150%) | 0x1c20 | Equivalent to 900µs length |
| Low position (-100%) | 0x2260 | Equivalent to 1100µs length |
| Neutral position (0%) | 0x2ee0 | Equivalent to 1500µs length |
| High position (100%) | 0x3b60 | Equivalent to 1900µs length |
| Extended high position (150%) | 0x41a0 | Equivalent to 2100µs length |

Channel data interpretation table

### Implementation of the SUMD Parsing

SUMD is a serial format and can be read directly from the receiver that’s connected via UART. Luckily Altera is offering an RS232 UART IP Core, which can be added to our SoPC using Qsys. It only requires two additional GPIO Pins, for receiving or transmitting serial data. Reading and controlling the UART will be part of the UART driver. The UART has to be initiated with the following settings, to receive a SUMD-Frame:

- 115200 Baud   
 - No Parity   
 - 1 Stop Bit

Every received Byte has to be interpreted according to the definitions of the SUMD signal format, which is described in the previous section. Following Steps are executed by the RC interpreted Controller:

### Saving raw SUMD-Frame Bytes from the UART

The SUMD-Controller has to wait for a new SUMD-Frame. A frame starts if the value of a received Byte equals the VendorID. After that, the following Bytes will be saved in an Array. The size of the Array will be equal to the frame this can be calculated with:   
SUMD-Frame length = SUMD Header length + Number of Channels \* 2 + CRC length)

### Interpreting the received SUMD-Frame

According to the SUMD format description, every Byte has its own specific purpose. The actual received RC-commands are sliced into a High Byte and a Low Byte, thus it is necessary to unroll both Bytes to a 16 Bit Integer. Every Channel value will be stored in an Array, which is accessible in a c struct including all additional Data of the SUMD-Frame.

## UART Driver

This driver will offer functions to initiate and read the RS232 UART IP Core. It is also possible to check if a new Byte was read. This is highly recommended if only one Byte will be read from the UART. The driver is divided in a source file "b\_uartriver.c" and a header file "b\_uartriver.h".

### SUMD-Frame-high

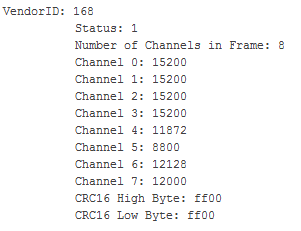


Figure SUMD-Frame-high

### SUMD-Frame-low

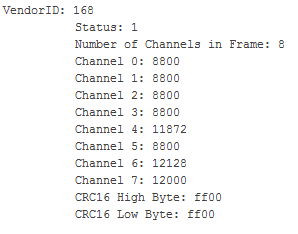


Figure SUMD-Frame-low

## PID Regulator

To guarantee a stable hovering, UAVs often employ PID regulators what was also done in the X-Copter project. The general functionality works like this: The PID has two input parameters and one return value. The input parameters are the set point and the real value whereby the set point is the value, the to-be-regulated-part (in our case one of the motors), should reach and the real value is the value, the motor currently has. These two values are subtracted and the resulting difference is known as the error. Now each part of the PID regulator manipulates the error respectively and returns it. The acronym PID stands for proportional, integral and differential and means that all of the parts are accumulated into one PID regulator. The factors have to be set individually for each project and setting by hand and trial and error. In the following these three parts of the PID regulator will be explained.

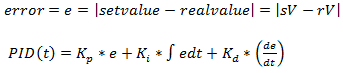


Figure 7 PID-Formula

### The Proportional Part

This is not a real regular for that matter. It's more a proportional amplifier that enforces the error by the factor. This is the most influencing part of the PID because it reacts very fast. For better understanding here is an example:

Let's say we want to have a motor run at 100 RPM. When we start the P-Regulator and the motor is not running the error will be

and it will return to the motor.

### The Integral Part

This is slowest part of a PID because it's an integral meaning its return value increases or decreases linear over time depending on the error being positive or negative. The bigger the positive errors are the faster the return value increases, the bigger the negative errors are the faster the return value decreases.   
For comparison to the P-Part: If the error is 0, the P-Part will return 0 but the I-Part return value won't change. So the I-Part serves mostly to correct the residual error of the P-Part.

### The Differential Part

The D-Part is the most nervous part of the PID because it reacts to temporal changes. So the faster the real value (or sensor data) changes the higher is the return value. This part does not depend on the RC signals. This is useful for fast response what is important in the system because it has to be stabilized at all the time. Let's say the XCopter is hovering stable and a wind gust is hitting the XCopter. The P-Part will recognize and steer against it in proportion to the error – but this to slow and it would move the UAV. The I-Part won't do anything helpful in this case because the time period is too short. The D-Part will detect a fast and big change of the real value so it will accelerate the motor very fast to correct the error and steer against the wind gust.

## Use of the sensor board of Sparkfun

For the stabilization of the model the 9 Degree of Freedom (9DoF) Stick from SparkFun is used. To verify orientation the stick provides an accelerometer, gyroscope and a magnetometer. Please refer to the final report of the previous group or to the datasheet of the sensor board [2] for exact information. Each sensor returns a value for the x, y, and z axis. Because each axis of the accelerometer and the gyroscope have a different range of values it is necessary to convert the output range into a uniform range. Only then it is possible to use the sensor values in the following modules of the flight controller. So the accelerometer and the gyroscope must be calibrated.  
[2] [3] [4] [5]

### Calibration of accelerometer

In contrast to the gyroscope the deviations of the accelerometer are invariable. For this reason calibration must be done just once. The results of calibration are an offset value and a scale value for each axis. This values are needed to convert the output range of the sensor into a uniform range.

To get the offset value and the scale value for each axis the maximum, minimum and the zero point values are needed. The How-to “Calibration of the accelerometer” [3] describes the procedure to get the values. The output data of the accelerometer is shown in the following chart (ref Output data of the accelerometer). The offset value is the result of the difference between zero and the measured value zero point.

[3] Path…../Calibration of the accelerometer.pdf UNAFFINDBAR!!!

Figure 9: Output data of the accelerometer

x

y

z

MAX

MIN

0

249

-13

-276

-264

-3

264

280

-227

26

The chart (ref Output data minus offset value) shows the range of the three axes after subtracting the offset value. To get the same range for all three axes a maximum value of 300 and a minimum value of -300 was defined. Therefore the scale value for the x axis is 300/262, the scale value for the y axis is 300/267 and the scale value for the z axis is 300/254. The offset value and the scale value are used in the “SensorDataFilter.c” component of the flight controller.

Figure 10: Output data minus offset value

x

y

z

MAX

MIN

0

262

0

-263

-267

0

267

254

-253

0

### Calibration of Gyro meter

A Gyro meter when started has does not start in a defined state, there is an error offset, especially if the sensor has been moved while it was switched of. Additionally the sensor has a warmup phase. In that phase the Values are invalid, they change over time even if the sensor doesn’t move.   
The calibration (measuring the offset) of the gyro meter needs to be done at every system start.  
To get the offset and warm up the gyro meter the Sensor Data Manager provides an init-method which takes samples and compares them until the difference between the samples is small enough then it saves the last sample as offset and the gyro meter is initialized.

# Lesson Learned //TODO 100 mal zu wenig

## Scrum

Scrum is a very good agile Software development tool. But for our team Scrum was a big overhead because of its planning for the far future. We needed too much time to build up User stories and tasks. In a big team of 9 people it was always a hard discussion until every person was clear. To maintain our Jira tool a lot of time was wasted. And we decided to continue without Scrum.

## Long term calendar management

Planning of team meetings was very hard because every person had a lot of work to do during the semester and it was hard to find a timeslot where everyone was available. Because we were discussing in small groups, communication problems occurred like the results and problems of the single groups wasn’t clear to all members.

## Small tasks take more time than thought

We thought some tasks could be done quickly. But in a hardware environment there is more than one thing to consider. So we needed more time to finish some tasks. In the future we are planning in smaller steps.