

FINAL REPORT

XCOPTER

XCopter Team T15
HOCHSCHULE ULM

1 TABLE OF CONTENTS

2	Project Description	5
3	Project Members	6
4	Analysis of Customer Needs	7
4.1	Previous Definition of Requirements:	7
4.2	Customer Needs	8
5	Project Management	9
5.1	Scrum	9
5.2	Roles in Scrum	9
5.2.1	Product Owner	9
5.2.2	Development Team	9
5.2.3	Scrum Master	10
5.3	Sprint 1	10
5.4	Sprint 2	12
6	Test Flights	13
6.1	First Test Flight	13
6.1.1	Attempt 1	13
6.1.2	Attempt 2	13
6.1.3	Attempt 3	13
6.1.4	Attempt 4	13
6.1.5	Conclusion	13
6.2	Second Test Flight	14
6.2.1	Organization	14
6.2.2	Conclusion	14
7	Interim Evaluation by the Team	15
8	System Architecture	16
9	Hardware	17
9.1	Power Supply	17
9.1.1	Solution for the Power Supply Problem	18
9.2	Remote Control	18
9.2.1	Choice of the RC – Controller and the Receiver	18
9.2.2	RC Transmitter Settings	19
9.3	Commercial Flight Control	20
9.4	Kinect and USB Host Controller	21
9.4.1	Initial Situation	21
9.4.2	Requirements for the USB-Controller	21

9.4.3	Common USB-Controller Packages	21
9.4.4	Selection of Suitable Controllers	21
9.4.5	Conclusion	23
10	Linux System	24
10.1	Reasons for Using a Custom Embedded Linux	24
10.2	Buildroot	25
10.3	Wi-Fi Connectivity	25
10.3.1	Implementing the Driver	25
10.3.2	Configure the Wi-Fi Connection	26
10.3.3	Result	27
10.4	MCAPAPI	27
10.4.1	MCAPAPI Entities	28
10.4.2	Packet Channel	29
10.4.3	MCAPAPI Example Program	29
10.5	MAVLink Communication Protocol	29
10.5.1	MAVLink on Linux	29
10.6	QGroundControl	29
10.6.1	Monitoring	30
11	Flight Controller	31
11.1	Flight Controller Basic Concept and Structure	31
11.1.1	What is a Flight Controller?	31
11.1.2	Basic Concept of a Flight controller	31
11.1.3	Structure of the XCopter Flight Controller	31
11.1.4	Dataflow of the Flight Controller	32
11.1.5	Programm Flow of the Flight Controller	33
11.1.6	Tasks and Timing	35
11.2	Motor PWM Signal	35
11.3	Interfaces Between Components	36
11.4	RC-Receiver	36
11.4.1	Graupner HoTT-SUMD-Signal Definition	36
11.4.2	Structure of a HoTT-SUMD Frame	36
11.4.3	SUMD_Header Section Description	37
11.4.4	SUMD Data Section Description	37
11.4.5	SUMD_CRC Section Description	37
11.4.6	Channel Data Interpretation	37
11.4.7	Implementation of the SUMD Parsing	37

11.4.8	Saving Raw SUMD-Frame Bytes from the UART	37
11.4.9	Interpreting the Received SUMD-Frame	37
11.5	UART Driver	38
11.5.1	SUMD-Frame-High.....	38
11.5.2	SUMD-Frame-Low	38
11.6	Sensor Data Manager	38
11.7	Sensor Data Filter	39
11.8	PID Regulators	40
11.8.1	General	40
11.8.2	PID Regulators in the Simple Flight Controller	41
11.9	PID to Motor Mapper	42
11.9.1	The Idea Behind the PID to Motor Mapper Module	42
11.9.2	Explanation: the Mapping Table.....	42
11.9.3	Explanation: Ensuring the Correct Boost of the PIDs	42
11.10	SparkFun Sensor Board	43
11.10.1	Calibration of the Accelerometer.....	43
11.10.2	Calibration of Gyroscope.....	45
11.11	Logging.....	45
12	Challenges and Problems	46
12.1	General Problems and Challenges with the Flight Controller	46
12.2	Project-Management	47
12.2.1	Time- and Team-Management.....	47
12.2.2	Scrum.....	47
12.2.3	JIRA	47
12.2.4	Git	47
12.3	Linux and Buildroot	48
12.4	USB-Controller and 3D-Mapping.....	48
12.5	PID and Motor Mapper	49
13	Lessons Learned	49
13.1	Project Management.....	49
13.2	Linux and Buildroot	49
13.3	USB-Controller and 3D-Mapping.....	49
14	Future Work	50
15	How-To Section	51
15.1	ARM/Linux: Virtual Machine, Cross Compilation, Executing code.....	51
15.1.1	Virtual Machine Usage and Folder Structure	51

15.1.2	Cross Compiling Code on Ubuntu for ARM	52
15.1.3	Connect to the ARM Linux System	53
15.2	Getting Calibration Parameters of the Accelerometer	54
15.3	Charging Batteries for the XCopter	55
15.3.1	Charging Batteries:	55
15.3.2	Step-by-Step-Solution.....	56
16	Bibliography.....	64

2 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 XCopter. Our goal is to continue to develop more software and test the flying ability of the XCopter. Another goal is to get the data while the XCopter is in the air. This will be achieved through an external monitoring-station on the ground.

3 PROJECT MEMBERS

- **Jan Goller**
 - Power Supply
 - Linux
 - Wi-Fi
 - Buildroot
- **Thomas Weber**
 - Flight Controller
 - RC Transmitter
 - RC Receiver
 - Sensor Calibration
- **Alexander Ott**
 - Flight Controller
 - Construction
 - Sensor Data Management
- **Florian Schneider**
 - Flight-Controller
 - Power Supply
 - PID and Filter
- **Daniel Maurus**
 - Linux
 - Wi-Fi and WPA
 - MCAPI
 - Accumulator
- **Stephan Gabor**
 - Linux
 - USB-Controller
 - Accumulator
 - MCAPI
- **Jochen Hoeft**
 - Pilot
 - Flight Controller
 - RC Receiver
- **Lukas Öffner**
 - Scrum Master
 - Communication Customer/Team
 - MCAPI and MAVLink

4 ANALYSIS OF CUSTOMER NEEDS

4.1 PREVIOUS 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 fiber 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." [1]

4.2 CUSTOMER NEEDS

Since this is an ongoing project, the basic hardware functionality and the inter-processor communication was implemented by the previous Team (Team Bumblebee). The customer needs of Team Bumblebee are documented in their Final Report ([1] page 5, "Analysis of Customer's Needs"). The current remaining customer needs are as follows:

Basic Needs:

- XCopter shall be a universal platform able to be fitted with a variety of sensors
- Stable power supply
 - Measuring the required power and guarantee a stable power supply that is sufficient to meet the needs of the system.
- Verification of Construction Stability
 - 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
- Choosing and commissioning remote control
- Remote controlled test flight with commercial flight controller

Logging and Sensors:

- Commissioning of sensors and evaluating sensor data
- Positional Tracking
- Automatic Compensation of Position in Space
- Logging of all sensor data (raw, filtered and PID-output)
- Choosing a fitting software solution for ground station
- Communication with ground station over the air
- Visualization of sensor data on ground station

Own flight controller software

- Implementation of own flight controller
 - The first version of the flight controller should offer the bare minimum functions to fly the XCopter model, however it has to be designed in a way that it can be given added functionality in the future. An example is would be to offer improvements to its stabilization capability by adding more sensors and 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".
- μ C/OS-II as real time operating system for NIOS II-CPU's (OS needed for MCAPI)

Future development

- 3D Environment Mapping with two 3D Cameras
 - A powerful external system for calculating 3D-data might be needed (mobile i7?)
- Autonomous Flight
- Collision control for the upper hemisphere (upper half of the XCopter)

5 PROJECT MANAGEMENT

5.1 SCRUM

The chosen software development methodology for this project is Scrum. It is an agile software development methodology for managing product development. The advantage of Scrum is, that it has an incremental and iterative approach. The basic unit in Scrum is named “sprint” or “iteration”. The duration of one sprint is usually between one week and one month and the sprints are performed iteratively. At the beginning of every sprint the Scrum team holds a sprint planning meeting. In this meeting the team selects the tasks which the team want to deal with in the following Sprint. All the tasks that can be chosen for the sprint are written down in the product backlog. The product backlog contains an ordered list of tasks and requirements that are needed for delivery of the final product. For example, the tasks can be features, bug-fixes or non-functional requirements. The product backlog is visible for every team member but can only be changed by the product owner. The product owner is also responsible for ordering the items of the product backlog in regards to the dependencies of the items and their priority. After the sprint planning meeting is finished the sprint starts and the selected tasks are assigned to the members of the development team. During the sprint the team meets to the “daily Scrum meeting” (in our case every week). The goal of the daily Scrum is, that every member of the Scrum team gives a short overview about the actual state of his task. It is necessary that every team member also talks honestly about issues that occurred during the time and don’t deny problems. The daily Scrum is just a short meeting and is usually limited to fifteen minutes. At the end of every sprint the Scrum team holds two meetings called “sprint review” and “sprint retrospective”. In the sprint review the team take note of which planned tasks are completed and which are not. At the end of the meeting there is a short demonstration of the completed tasks for the stakeholders. This is also the point where the stakeholders have the chance to comment the results of the Scrum team or to talk about improvements. The following meeting is the sprint retrospective which is without the stakeholders. In this meeting the Scrum team talks about the previous sprint and the problems that occurred during this sprint. Here the team can also talk about personal problems like bad communication between different team members or missing resources. The goal of this meeting is to improve procedures for the following sprint. After the sprint review, the next iteration starts again with the sprint planning meeting. One of the key principles of Scrum is its ability to implement customer changes during a project. Scrum is a step by step developing methodology and that is why it was decided to use Scrum. [2]

5.2 ROLES IN SCRUM

5.2.1 Product Owner

The product owner is closely in contact with the stakeholders and arbitrate between the customer and the Scrum team. The Scrum master just looks after the business side of the project but not after the technical aspects of the product. He writes customer-centric items typically, the user stories, ranks them and prioritizes them. In the XCopter project the product owner is same as the customer. [2]

5.2.2 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 has their own tasks. In each task the actual work is described by the product owner and Scrum master. If the development team finishes tasks, they continue with the next open task. [2]

5.2.3 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 facilitates team events like the daily Scrum or other meetings. He acts as a buffer between the team and the customer. [2]

5.3 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. It can be seen in the Burndown Chart below in Figure 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. [2]

1. **SELECT THE REMOTE:**

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

2. **STABLE BATTERY**

- getting the circuit diagram
- checking the existing board
- building the circuit on a prototype board

3. **CHARGING THE BATTERY**

- programming the charger
- extern power supply

4. **USB-CONTROLLER**

- comparison of USB-Controllers
- requirements
- searching for new USB-Controller

5. **COMMERCIAL FLIGHT CONTROLLER**

- getting the Software and installing it
- feature list
- configuring the cruise control
- connecting the cruise controll to the rotors
- connecting the cruise controll to the flight controller
- connecting the flight controller to the XCopter
- getting a connection to the remote
- configurating the flight controller
- configurating the remote

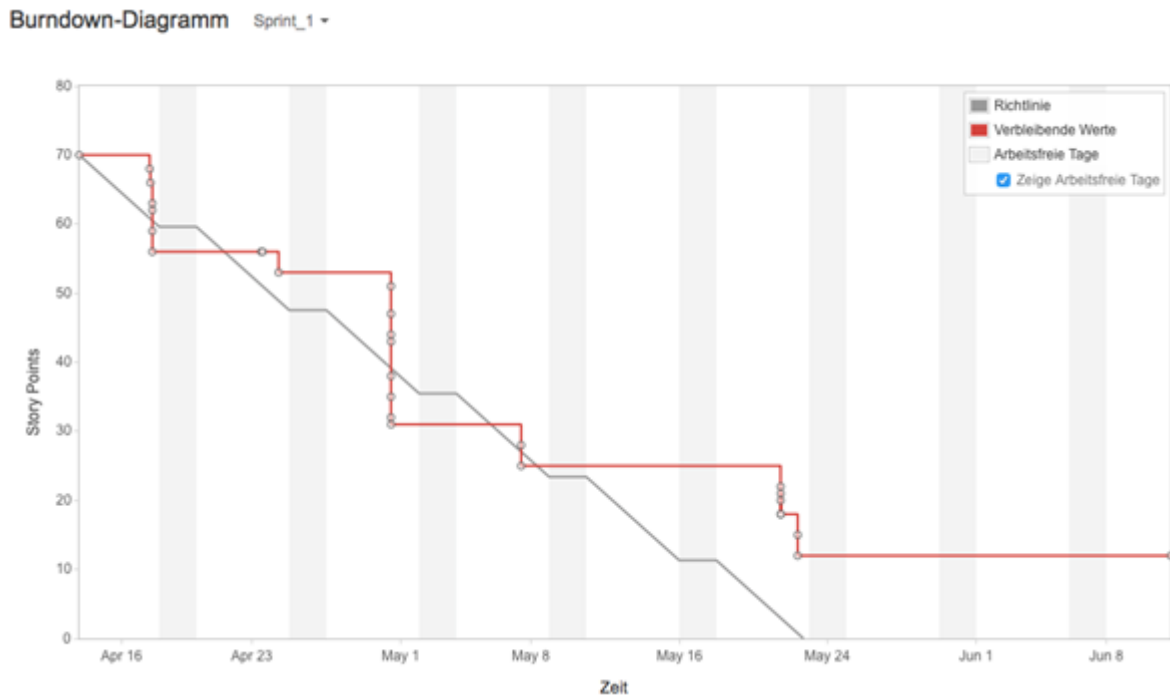


Figure 1 Burndown-Diagram Sprint 1

The Burndown-Chart shows how the team completed the list of tasks that was created in the Sprint meeting. It is taken from the JiraWeb tool where the team organized all Scrum activities. The X-axis describes the time from the start of the Sprint to the end. One Sprint lasts one month. The Y-axis represents the storypoints of each task. A story point starts from zero in a scale of up to five. Five story points mean that the task is very hard and needs a lot of time. As seen in the Burndown-Chart, the team handles most of the tasks in one month. In the middle of May, there was a task that needs more time than expected. It was the USB-Controller which wasn't completed in the first Sprint. Eventually the first assessment was very good and most of the tasks were finished at the scheduled time.

5.4 SPRINT 2

1. Commercial flight controller
 - Backing up the configuration of the controller
 - Getting a date for the first flight
 - Organizing a bus for the test flight
 - Switching for the RC-Controller
 - Building a frame for landing
 - Adding fuses for the XCopter
2. Simple flight controller
 - Getting information about the drivers
 - Information gathering about components of a flight controller
 - Getting information about PID regulator
 - Testing the drivers
3. USB-Controller
 - Comparing suitable USB-Controllers
4. Day of informatics
 - Making a presentation about the XCopter
 - Making a poster for the day of informatics

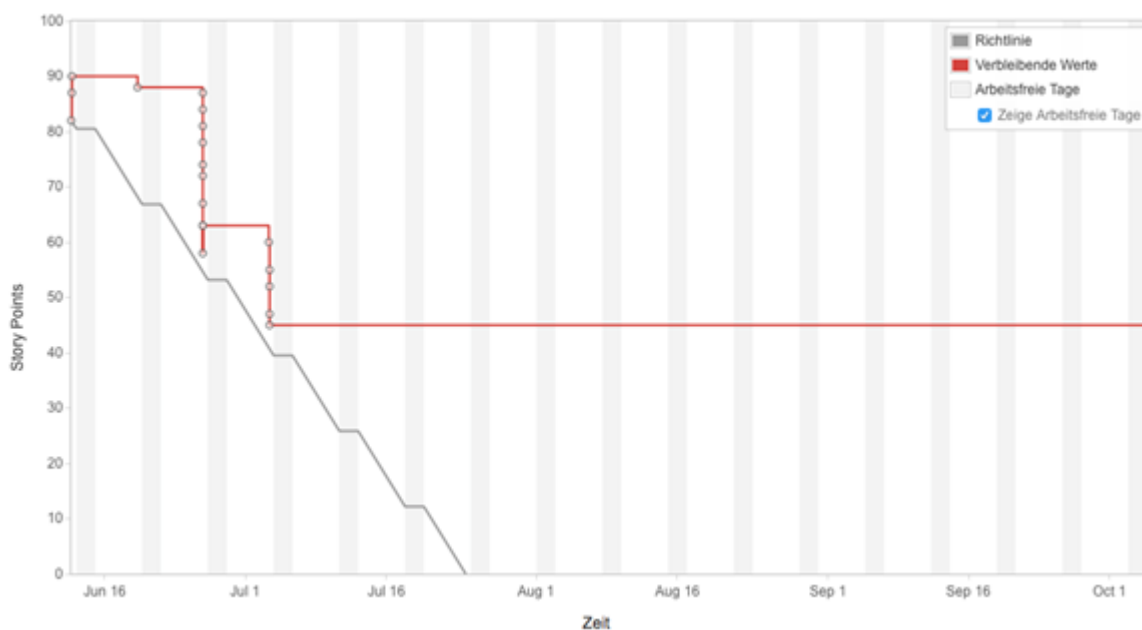


Figure 2: Burndown-Chart Sprint2

The second Burndown-Chart shows nearly the same as the Burndown-Chart for Sprint 1. The team completed some tasks in the first month. After this the team had problems with the USB-Controller again and also with some other tasks that needed more time than expected

6 TEST FLIGHTS

6.1 FIRST TEST FLIGHT

For safety reasons it was not possible to test the XCopter in the laboratory or at the campus. So for testing it was necessary to go to a model flying site. In addition the person that controlled the multicopter had to have a model flying insurance. Jochen Höft already had a flying insurance and connections to the near model flying site in Staig, the first flight of the multicopter was done there. At first all components got attached to the model. In addition to the aircraft, basic equipment of tools and spare parts was taken to the model flying site.

6.1.1 Attempt 1

The first attempt failed. The motors had the wrong direction of rotation. So there was no upwards boost but a downwards boost.

Error analysis: Falsely it was assumed that the rotors were mounted incorrectly. So the rotor were switched which led to upward boost from every rotor.

6.1.2 Attempt 2

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

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

6.1.3 Attempt 3

This attempt started well but the XCopter lost upward boost at one side and crashed.

Error analysis: It turned out that one rotor loosened. All rotor screws were tightened.

6.1.4 Attempt 4

The take-off went well. However, one motor flew away.

Error analysis: The rubber vibration damper that was used to mount the motors seemed to be not strong enough. The tools and spare parts were not enough to mount the motors without the vibration damper.

6.1.5 Conclusion

The XCopter vehicle should have been able to fly. The rubber vibration damper is not recommended, because the motors apply to much strength on them and finally could cause them to break (Figure 3). Another test was necessary. Before a flight it's necessary to check the mechanical stuff as well as the electronic components for their correctness. Therefore a simple checklist was made to save a lot of time. All in all it was despite the problems an achievement, because the chassis of the XCopter is very solid and in principle the vehicle could flight.

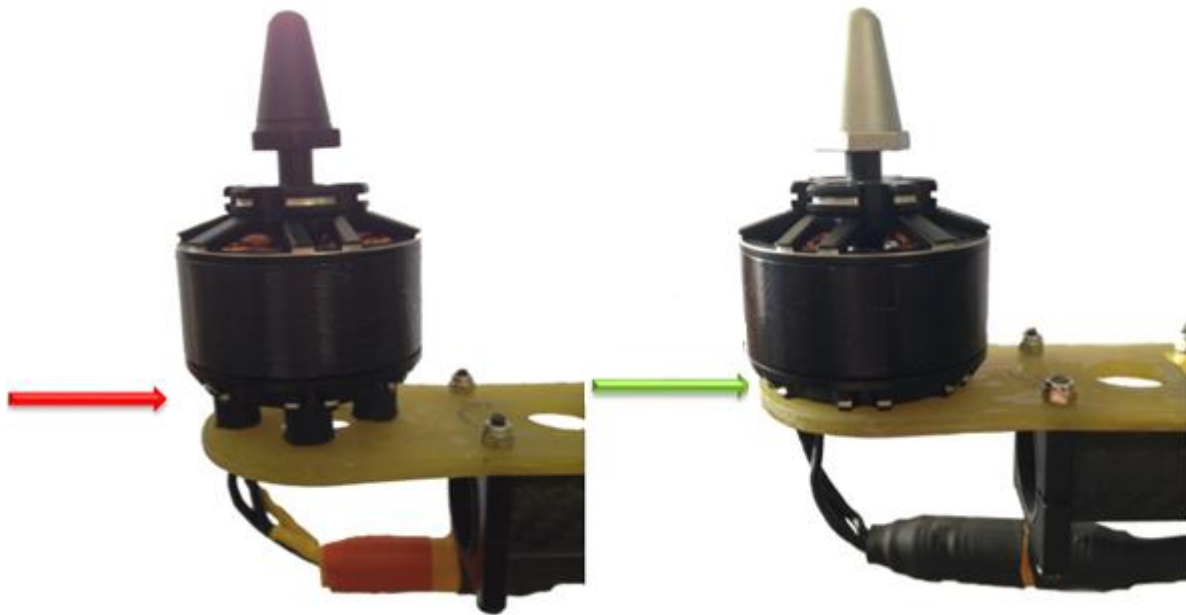


Figure 3 Damper improvement

6.2 SECOND TEST FLIGHT

6.2.1 Organization

The organization 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. The preparation of the team was much better, a checklist was made to avoid previous mistakes. In one part the XCopter is different. 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. Even though the conditions were bad, it was very windy this day, the XCopter flew very well. Without any experiences about the maximum flight time, the team decided after 10 minutes to land the XCopter. The test was a success.

6.2.2 Conclusion

In quadcopter form the XCopter 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 in principle without any further modifications. With the experience of the first test flight and the different failed attempts it was no problem to get the XCopter into the air. In the future it could be possible that the cameras, which should be mounted to map the surroundings, have to be fixed with dampers in case of too much vibration. In other self-made multicopter projects this is the way to do it, so there should be no problems with it. The flights were very important to set the basis for further procedure of the project. In further steps, the team can concentrate on the main aspects of the project.

7 INTERIM EVALUATION BY THE TEAM

After half of the project some important milestones, like the first successful flight or the successful transmission from the RC transmitter to the NIOS II-processor were reached. 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 tasks in a short time. However, at some tasks, the team 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 the chosen development methodology. Scrum was a loss of time. The team decided to focus on the software development and not on Scrum anymore. The software development methodology was customized and the team decided to do a kind of downgraded Scrum. Main tasks were defined by the whole team and allocated to smaller teams. All team members should know about the tasks the others do, but not in detail. All the small teams have to organize themselves in case of time slots and dividing the big tasks into smaller ones. Also some improvements in the organization and the project structure were done. Thus we should be able to get more work done, in a shorter period of time in order to reach our goal. More information about problems with Scrum can be read in section 12.2.

8 SYSTEM ARCHITECTURE

The preliminary findings in the power supply and USB topics, lead to some changes in the system architecture, compared to the system architecture when the XCopter project started.

At first, the old power supply circuit on the extension board was replaced by a commercial ready-made and extern voltage regulator, which can be placed somewhere on the XCopter (see chapter 9.1). A second change contains the 3D cameras. It was proposed to implement two USB-Host controllers on the extension board to connect a Kinect camera to each of them at the beginning. The insights that were gathered when working on the USB-Host controllers led to another solution (see chapter 3). The Kinect cameras might be replaced by two ASUS 3D cameras that would be connected to the USB controller on the DE1-SoC board. Before the ASUS 3D cameras can be used in the XCopter, a validation has to take place.

It was also decided to connect the XCopter to the ground station, which is running on a PC or Laptop, via Wi-Fi. Therefore a USB Wi-Fi module was used and connected to the DE1-SoC Board (see chapter 10.3).

The other architecture specifications didn't change. The DE1-SoC Board contains three CPUs, an ARM9 dual core that handles the coordination, and two NIOS II CPUs. One of them is responsible for collision detection and the other CPU for the flight controller. Each of those CPUs communicate with the other processors via MCAPI (see chapter 10.4). An accumulator pack powers the motors and the DE1-SoC Board. Because the board needs 12V, the voltage regulator converts the accumulator potential before. The extension board holds the orientation sensors, that are necessary in the flight controller, as well as the distance sensors for the collision detection. To catch the remote control commands, an RC receiver (see chapter 11.4) is connected to the extension board too. The motor controllers, to which the motors of the XCopter are connected, are also plugged to the extension board. To make the development of the flight controller easier, there are only four of the six possible motors connected at the state of the project when this text was written.

More details on the specific components can be found through this document at the respective chapters (see chapter 11).

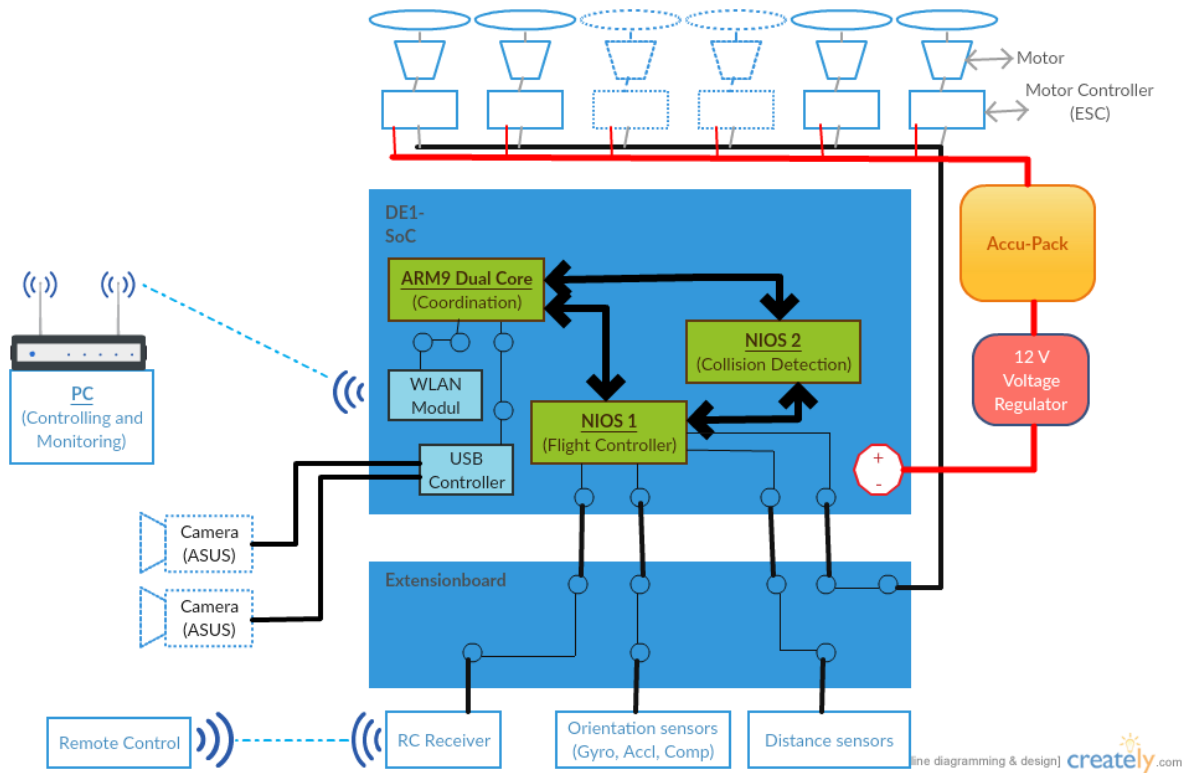


Figure 4: System Architecture of the XCopter

9 HARDWARE

9.1 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. 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. 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 with the XCopter and the SoC-Board's restrictions. The restrictions are 12V input voltage and 3.5A input 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 a problem with the plan itself.

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 created two circuit boards, used different ICs on each of them. It was concluded that at least one of the boards cannot work.

Big error sources are the SMD parts. The problem with them is that they cannot easily be tested. Therefore all the parts were bought in DIP norm and the circuit was built on a plug board. 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 us missing experience and tools on this subject, it was decided to buy a commercial power supply from an online shop (see Figure 5).

9.1.1 Solution for the Power Supply Problem

The solution is a DC to DC voltage regulator which fulfills all the necessary parameters. 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



Figure 5: Commercial power supply

9.2 REMOTE CONTROL

9.2.1 Choice of the RC – Controller and the Receiver

The XCopter is designed to be controlled by a conventional RC Transmitter. A more specific explanation why the Graupner MX-16 HoTT was chosen, is listed below.

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

- Support of sum signal (PPM)
- Providing 4 channels or more, 8 channels are optimal
- The costs have to be less than 350€
- Easy configuration of the RC- Controller

A selection of leading companies producing RC- Controllers:

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

Reasons for Graupner:

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

Major properties of the RC-Controller:

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



Figure 6: Graupner MX16 [3]

9.2.2 RC Transmitter Settings

This section is about how the Graupner HOTT receiver has to be set up. Every option listed below is a translated version of the option entry, since the language of the transmitter is German. The German option names are singed as [Name]. The XCopter profile settings on the MX-16 transmitter is described in the following section.

Control mode [Steueranord.]	2
Throttle stick behavior [Motor an K1]	"kein" (default)
Changing flight phase with channel 8 delayed [K8 Verzögert]	"nein" (default)
Tail unit type [Leitwerk]	"normal" (default)
Tail unit servo type [Querr./Wölb]	"1QR" (default)
Channel Output [Empf. Ausg]	S 2 -> [Ausgang] 1 S 3 -> [Ausgang] 2 S 1 -> [Ausgang] 3

Table 1: Basic model settings [Basic settings]

Receiver output type [CH OUT TYPE]	"SUMD HD 08"
------------------------------------	--------------

Table 2: RC Receiver Settings [Telemetry]

More information is available in the original Graupner MX-16 user manual ([4]). Graupner HoTT can deliver a digital SUM-signal (SUMD more in section 11.2) on most common HoTT receivers. The signal can be found on the following outputs:

- GR-12L -> Output #6
- GR-16 -> Output #8
- GR-24 -> Output #8
- GR-32 -> Output #8

9.3 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, we decided to install a commercial flight controller. 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 multicopter 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.

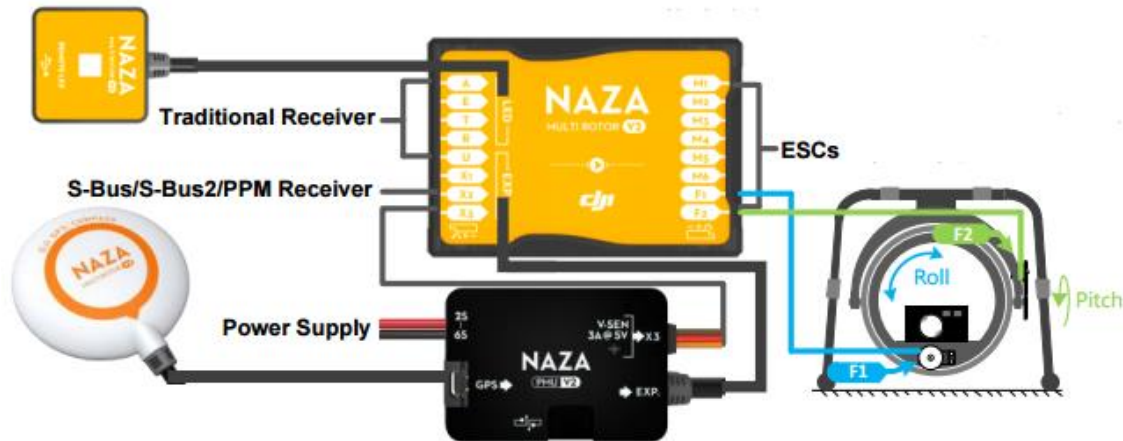


Figure 7: Commercial Flight Controller [4]

9.4 KINECT AND USB HOST CONTROLLER

9.4.1 Initial Situation

The customer wants two Kinects to be put on the XCopter. They are to provide 3D-Image data to map the surrounding locale of the XCopter. 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 color camera. For proper 3D-image, data color- 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 a 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 [5].

9.4.2 Requirements for the USB-Controller

There are certain cut in stone requirements for the USB-sontrroller 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 solderable
- Full High-Speed data transfer rate of 480 MBit/s
- (Should be ULPI compatible if present Waveshare 3300-transceivers from the BumbleBee-group are meant to be used)

9.4.3 Common USB-Controller Packages

There are three different common USB-controller packages that are solderable 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 ([6]). QFN is harder to solder which is why QFP style packages are the preferred choice.

9.4.4 Selection of Suitable Controllers

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 led 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. Even further investigation showed that the

successor to the ISP1761BE, the SAF1761BE from NXP Semiconductors, is also available to purchase and is 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.

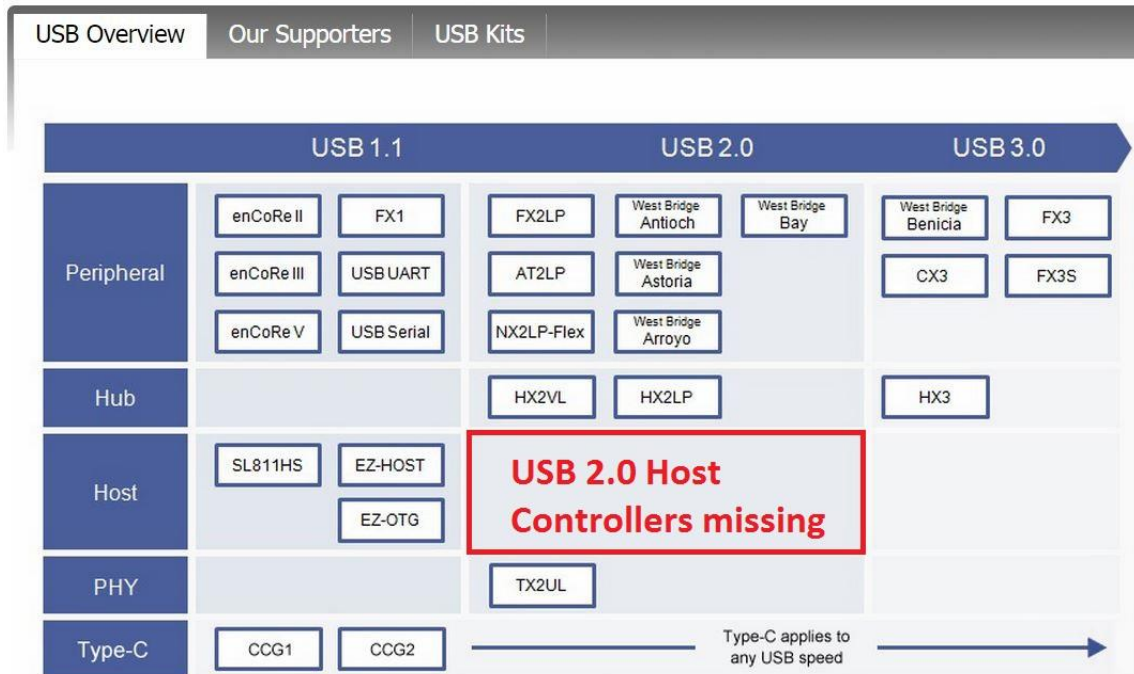


Figure 8:Cypress USB offerings [6]

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	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

Table 3 USB controller comparison

9.4.5 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 for the common consumer market.

Which leads to the FT313H(L/P) which offers the best characteristics for our endeavors. It is still supported and offers Linux drivers. It has a relatively low footprint, is solderable with the tools at hand and comes in two packages: 64LQFP and 64TQFP. It supports High-Speed USB 2.0 transfer rates and can also be ordered with a development module.

10 LINUX SYSTEM

10.1 REASONS FOR USING A CUSTOM EMBEDDED LINUX

There are many Linux distributions that are designed to specifically run on ARM devices. When thinking about what OS to run on the ARM part of the DE1-SoC system, the question arises if a pre-built OS, or a custom built OS should be chosen. Choosing a pre-built OS would have been the way of least resistance when considering the OS-question, especially as Altera has pre-configured OS-images on their website to download. A custom built OS on the other hand has several advantages:

- **Customizable**
 - The fact, that a custom built OS is more customizable, than a pre-built OS is self-evident and doesn't need to be explained in more detail.
- **Light weight**
 - Light weight means, that the operating system can be stripped from any functions that are not needed for what the system is intended to do. Functions can be added for development, debugging and testing for a better experience while working with the OS. Added features can then be removed from the custom distribution in a later stage of the project when everything is set up correctly and the system is running stable. In the case of the XCopter, the final OS can be stripped from anything that is not related to logging sensor data, communication with the ground station and eventually 3D mapping in the future.
- **Performance**
 - Light weightness brings also the benefit of better performance. It is obvious, that a stripped down system is working much faster on its delegated tasks, when there are no unneeded services running in the background that use up CPU-power.
- **Security**
 - Security is an important point to consider when designing UAVs. With a stripped down OS there are less attack vectors for intruders and the system should be more secure overall.
- **Learning effects**
 - Custom built operating systems are widely used in modern embedded design and being able to work with such a system on the XCopter project yields great opportunities to have hands on experience with an embedded OS.

The biggest disadvantage is the steep learning curve for working with custom embedded Linux distributions, especially for first timers. It is also a big challenge to get additional hardware like Wi-Fi-Sticks working as drivers sometimes have to be customized, or even written from scratch, to work with the system. Sometimes drivers and support for specific hardware is already implemented into a toolchain for building the embedded operating system (like Buildroot or Yocto). But even if drivers are present in the toolchain, finding them in convoluted menus and choosing the right dependencies (like the correct eeprom support etc.) is time consuming. For our project the Buildroot toolchain was chosen for generating and configuring the real-time embedded operating system.

A stable Kernel with version number 3.10-ltsi-rt (custom repository version) is used for the embedded Linux. It is not important to have the newest Kernel that is currently on the "Kernel-market", but it is of importance to have a stable and supported version.

10.2 BUILDROOT

Buildroot is an open source project that makes it possible to create an individual Linux system. It is divided in several 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 limited resources. The easiest way to implement new drivers into the ARM Linux system is to do it with Buildroot. Cross-compiling drivers is very difficult and can lead to a lot of errors. Even if there is a driver available for an ARMv7 architecture (which is the architecture that our system uses) it doesn't mean that it will work out of the box.

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 bootloader. Its task is to boot up the Linux system. Usually these boot loaders have 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 were built by Mr. Strahnen. The system runs on kernel 3.10.37, because of the long term support. It is not possible to compile code on the ARM CPU because of the missing toolchain. Therefore it is necessary to cross-compile the code on the virtual machine (15.1). Components like the Wi-Fi driver were added via Buildroot. To increase the performance of the system it is possible to deactivate the unused packages. [7] [8]

10.3 WI-FI CONNECTIVITY

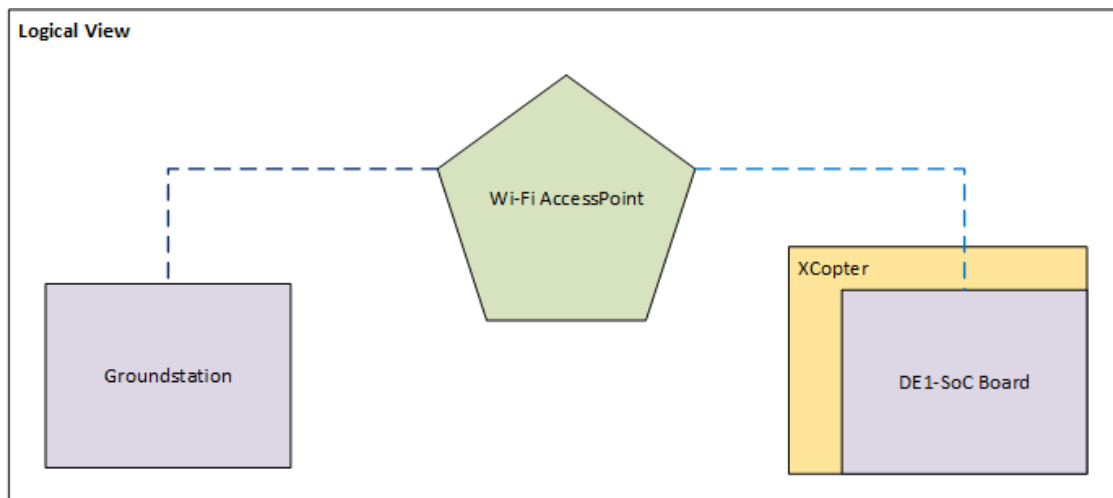


Figure 9: The logical view of the Wi-Fi components

The XCopter 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 (from Edimax with an RT18188 chip) plugged in the XCopter DE1-SoC board. Both are connected with an access point (Figure 9: The logical view of the Wi-Fi components).

10.3.1 Implementing the Driver

First step to implement the driver was to establish a connection between the DE1 and the access point. The Wi-Fi dongle doesn't work out of the box. To get the dongle working there are two possibilities. First one is to compile a Linux driver and the second is 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 is 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.

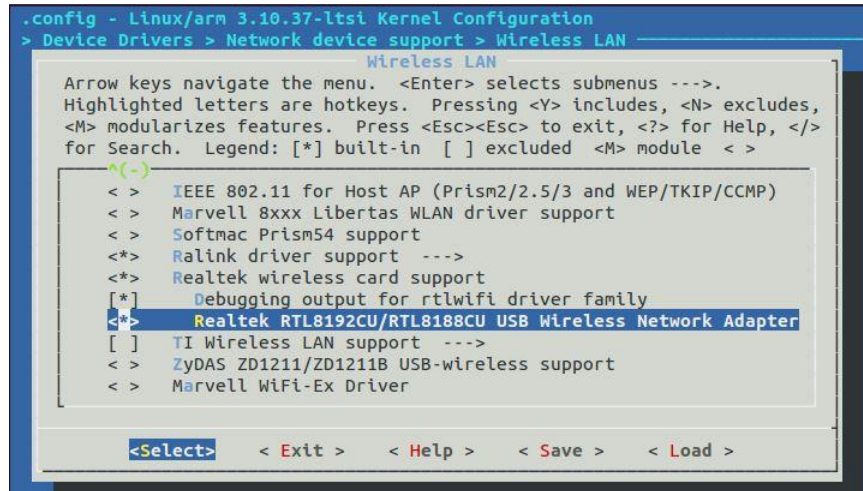


Figure 10: Buildroot driver selection

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 (Figure 10: Buildroot driver selection). Now the driver file can be loaded successfully but there has been an error with a missing firmware file. Adding the right firmware in Buildroot solved this issue. Now the basic settings are set and the DE1 can establish a wireless connection to the access point.

10.3.2 Configure the Wi-Fi Connection

To configure the network there are two files to edit. One is the `/etc/network/interfaces` and the other is the `/etc/wpa_supplicant`. In the interfaces file following changes have to be set up (Figure 11: Interface configuration file):

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

# Wireless interfaces
iface wlan0 inet dhcp
    wireless_mode managed
    wireless_essid any
    wpa-driver wext
    wpa-conf /etc/wpa_supplicant.conf
```

Figure 11: Interface configuration file

The `wpa_supplicant` file (see Figure 12: `wpa_supplicant` file) needs the following entries to work.

```
ctrl_interface=/var/run/wpa_supplicant
#ap_scan=1

network={
    ssid="FlightNet"
    psk="12345678"
    scan_ssid=1
    proto=RSN
    key_mgmt=WPA-PSK
    group=CCMP TKIP
    pairwise=CCMP TKIP
    priority=5
}
```

Figure 12: wpa_supplicant file

To get an IP lease from the access point it is necessary that the daemon *wpa_supplicant* is running. To start the background process, type the command:

```
/usr/sbin/wpa_supplicant -I wlan0 -D nl80211 -c /etc/wpa_supplicant.conf
```

into the terminal. It is recommend to let this automatically run at the start up. A new file was made in the */etc/init.d/* folder with the file name *S50start_wlan* and inserted in the content of Figure 13: Content of the start-up script.

```
# wlan0          Starts wpa_supplicant.
#
start() {
    echo -n "Starting WLAN0 "
    /usr/sbin/wpa_supplicant -i wlan0 -D nl80211 -c /etc/wpa_supplicant.conf
    sleep 8          #wait 10 seconds to finish wpa_supplicant job
    ifup wlan0       #establish wifi connection

    echo "OK"
}
stop() {
    echo -n "Stopping WLAN0: "
    killall wpa_supplicant

    echo "OK"
}
restart() {
    stop
    start
}
```

Figure 13: Content of the start-up script

10.3.3 Result

The hardware driver is included and the Wi-Fi connection is established. At booting Linux, the *wpa_supplicant* daemon is starting and the IP leases automatically. The connection is encrypted and is secured for non-authorized access. The Wi-Fi connection was successfully tested with MAVLink protocol. The ground station QGroundControl was running on a notebook and on our portable system-on-a-chip device. The cross compiled heartbeat test application was sending a continuous proof of life over the air.

10.4 MCAPI

MCAPI (Multicore Communications API) is a standard which allows to communicate between processes, running on different CPUs. The XCopter has three separated CPUs. The Linux runs on the ARM CPU and the flight controller is running on the NIOSII s0 (For a more detailed look of the system architecture see

chapter 8). The NIOSII s0 CPU, which holds the flight controller, sends the logged telemetry data to the ARM CPU. MCAPAPI is necessary for the communication between the CPUs.

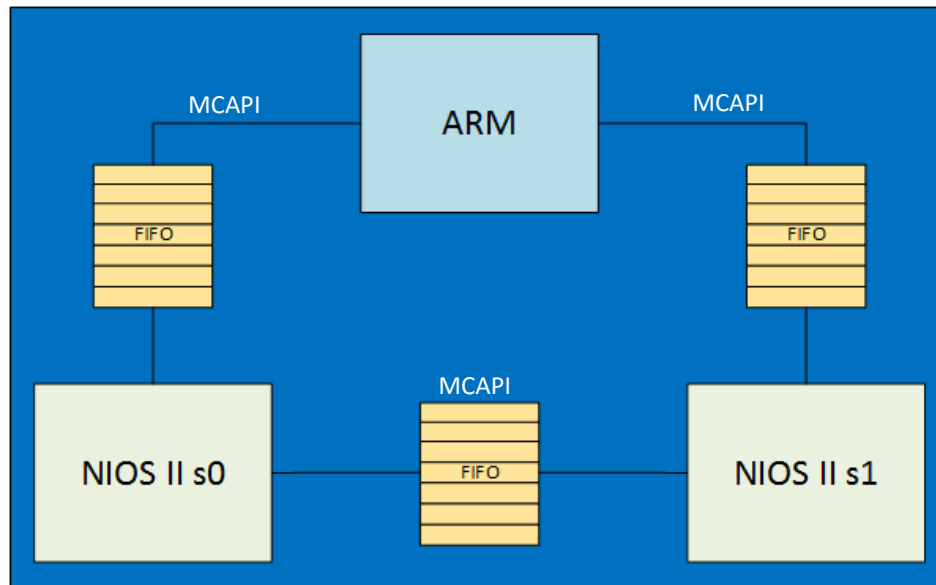


Figure 14 MCAPAPI communication overview

In the following, the MCAPAPI basics are explained.

10.4.1 MCAPAPI Entities

For the MCAPAPI there are three basic entities: domains, nodes, and endpoints.

Next a short description of the tree entities:

Domain

A MCAPAPI domain gathers several MCAPAPI nodes. Domains can be nested.

Node

A MCAPAPI node is in general an independent unit of control. But every MCAPAPI implementation has to specify its definition of a node.

Endpoint

A MCAPAPI endpoint is a socket like communication structure. An endpoint must be unique. It is given by a tuple of domain-ID, node-ID and port-ID. One endpoint belongs to one node. But a node can handle several endpoints.

10.4.2 Packet Channel

In the XCopter project, the packet channel mode was chosen. It is a bidirectional communication between the endpoints. There are a receiving node and a sending node.

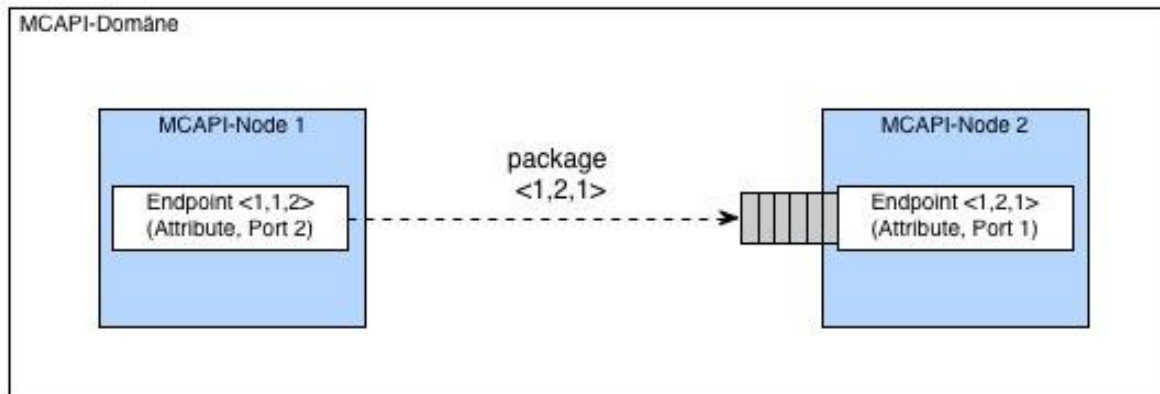


Figure 15 Packet channel data transfer

10.4.3 MCAP Example Program

In the VM there is a folder I_MCAPI. There are two C files, one for the NIOS and one for the ARM side. Every program does an initialization of the channels. This is all done in the initialization method. After this initialization code the application sends and receives data in a while loop. This is the entry point for future work. There it is possible to send data from the NIOS to the ARM CPU and to display log data on the ground control station.

10.5 MAVLINK COMMUNICATION PROTOCOL

MAVLink is a very lightweight, header-only message library for micro air vehicles. It can pack C-structs and send them efficiently over serial packets to the ground station. MAVLink was first released 2009 by Lorenz Meier under LGPL licenses.

10.5.1 MAVLink on Linux

To send data from the ARM processor to the ground station, the example program for Linux integration on the MAVLink website [9] was used. This program tests the UDP connection to QGroundControl. The program sends the necessary MAVLink packets to the QGroundControl which answers with a heartbeat. The example program can be compiled on Linux without changes. The only thing that had to be changed was the IP address from localhost to the IP of the target PC where QGroundControl was running.

10.6 QGROUNDCONTROL

QGroundControl provides full support as a ground station. It supports all kind of vehicles that can handle MAVLink. Here is a short list of the main features:

- Open-source
- Windows / Linux / MacOS support.
- 2/3D aerial maps
- In-flight manipulation of waypoints
- Real-time plotting
- Logging of sensor logs
- Supports multiple autopilots
- MAVLink protocol support

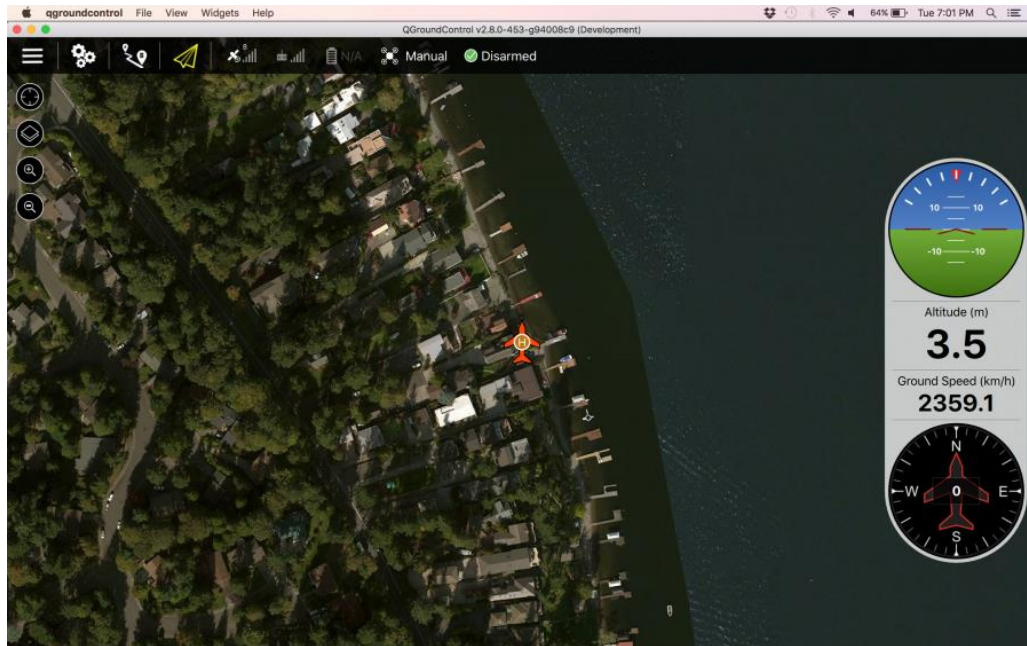


Figure 16 QGroundControl [10]

10.6.1 Monitoring

Combining all the communication protocols, there is a chain that the XCopter is used for monitoring. Its starts by logging data on the NIOS side, like values of the sensors etcetera. To visualize this data the NIOS CPU sends its data to the Linux via MCAPI. The Linux packs this data in a MAVLink struct and sends his packet over a socket to the QGroundControl. QGroundControl visualizes the data and monitors it on any PC that runs a connected instance of QGroundControl. The following chart shows the communication chain for a better comprehension:

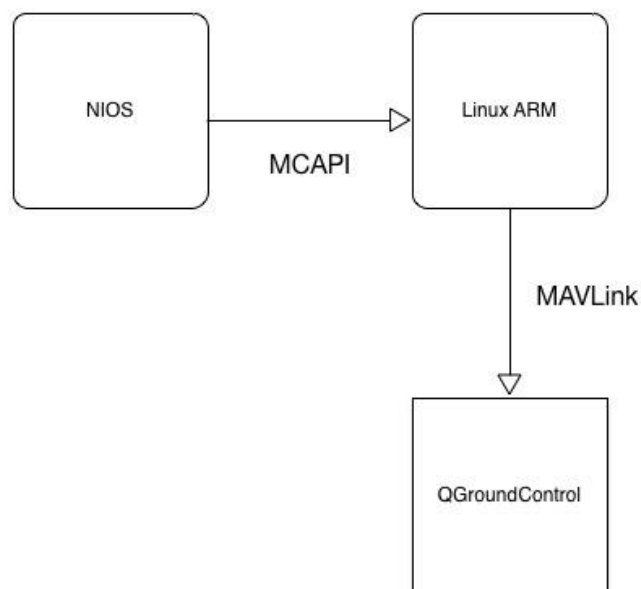


Figure 17 QGroundControl communication chain

11 FLIGHT CONTROLLER

11.1 FLIGHT CONTROLLER BASIC CONCEPT AND STRUCTURE

11.1.1 What is a Flight Controller?

The heart of every UAV is the flight controller. This unit is responsible for keeping the aircraft stable in the air while executing the commands, the UAV gets from the user via the remote control (RC). There are several commercial flight controllers available that could be mounted on the XCopter as described in section 9.3. For the purpose of this project an own, non-commercial flight controller is implemented. It's basic idea or concept and structure as well as the dataflow is described in the following subchapters and figures.

11.1.2 Basic Concept of a Flight Controller

For every flight controller the basic concept is more or less the same: The UAV has sensors such as a gyroscope, an accelerometer and (sometimes) a compass on board. The main control loop does the following as long as the UAV is in the air: With the data of the sensors, the pitch, yaw and gear orientation (see Figure 18) of the UAV is calculated.

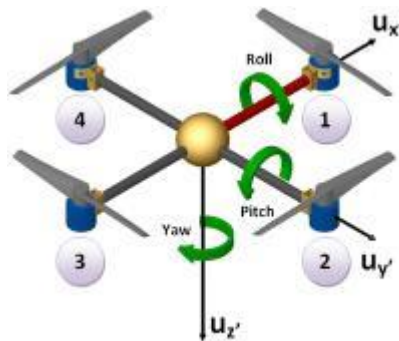


Figure 18: Roll, Pitch and Yaw orientation [11]

The actual orientation and the desired position, which is transmitted by the user via the remote control, are the inputs for the PID regulators. (For a description of what a PID is, see 11.7. Depending of the output of these PID's the individual motors have to run at different speeds to stabilize the UAV in the air.)

11.1.3 Structure of the XCopter Flight Controller

In the XCopter project, the flight controller is separated into three tasks. One task handles the remote control input, the second task is responsible for gathering the sensor data, and the third task holds the main controlling algorithm. The three tasks are synchronized and share their data with mutexes and semaphores. The detailed properties and timings of the tasks are described later in section 11.1.6. The program flow can be seen in more detail in the chapter 11.1.5.

In Figure 19 a component diagram, that shows the structure of the flight controller, can be seen. It should be noted that the position of the components has no meaning, it's just for a better readability. All the hardware drivers, which are providing the interfaces to communicate with the sensors and motors, are colored in white. The three parts mentioned above are highlighted with different colors. The Sensor Data Manager is colored in pink and its details are described in section 11.6. The RC receiver is colored in green and is elucidated in section 9.2. The main controlling part contains multiple components and is colored in yellow. Its particular components are described separately in later sections, whereas the functionality will be pointed out in the following section.

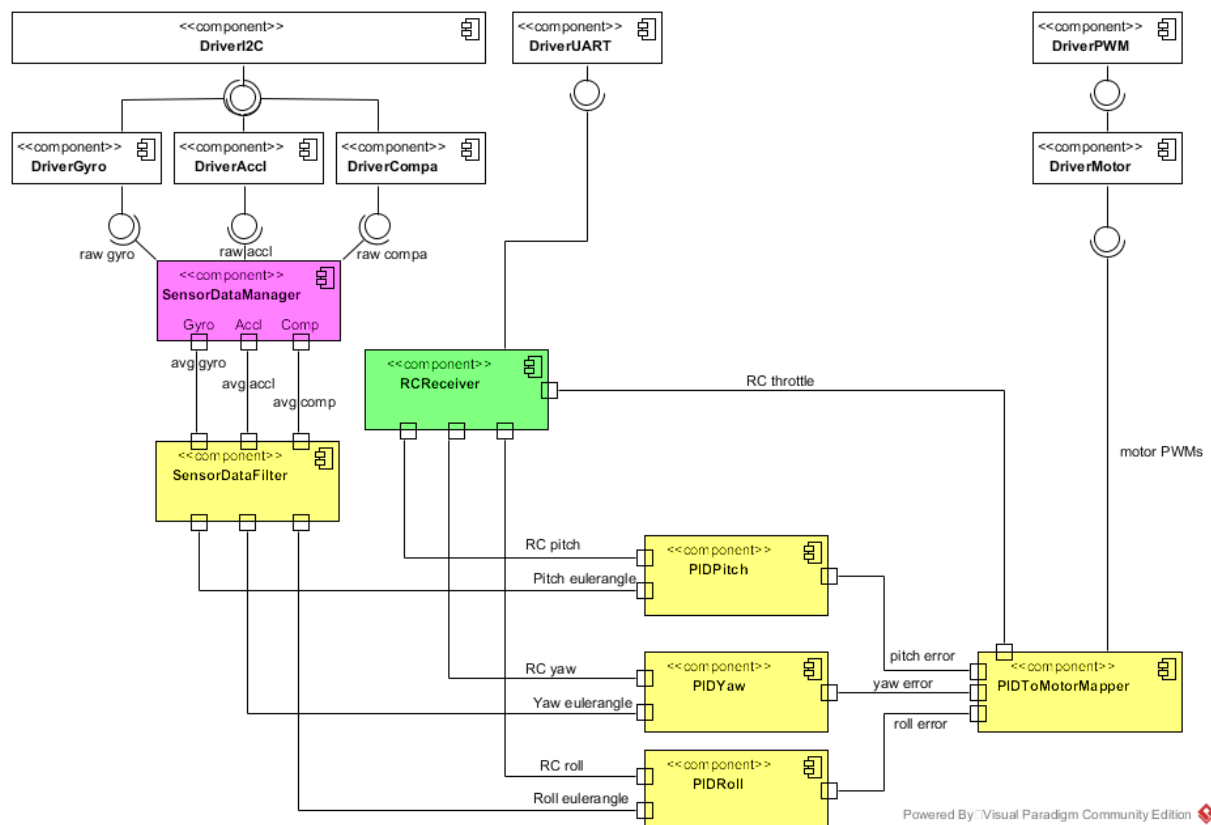


Figure 19: Flight Controller structure

11.1.4 Dataflow of the Flight Controller

Basically, what a flight controller does, is getting data from the sensors and the remote control and translate this into engine speed. Or the other way round: The motors are running at different velocities depending on the data they are given from the user (remote control) and the sensors. Therefore, understanding the dataflow is very important because it is also a big error source. In the following, an overview of the dataflow in the XCopter flight controller is given. For more details of the data, e.g. the ranges in which the values are, refer to section 11.3. More details about the different components can be found in their respective chapters.

In Figure 20 an overview of the dataflow through the flight controller can be seen. At first, the Sensor Data Manager collects the raw data from the accelerometer, the gyroscope and the compass. Then the raw data gets averaged and handed over to the Sensor Data Filter. This component then filters the data and computes the current orientation of the XCopter as Euler angles (pitch, yaw, roll) by combining the data from the sensors.

When the Euler angles are calculated, the second data source, the remote control comes into play. The remote control also provides Euler angles depending on the orientation of its joysticks. The Euler angles from the Sensor Data Filter and the remote control are then fed into the particular PIDs. Each of those PIDs computes an error value for each axis.

The throttle value, which also comes from the remote control, and the PID error values are the input of the PID to Motor Mapper. This component translates the error values in PWM signals that are then mapped on the motors by the motor drivers.

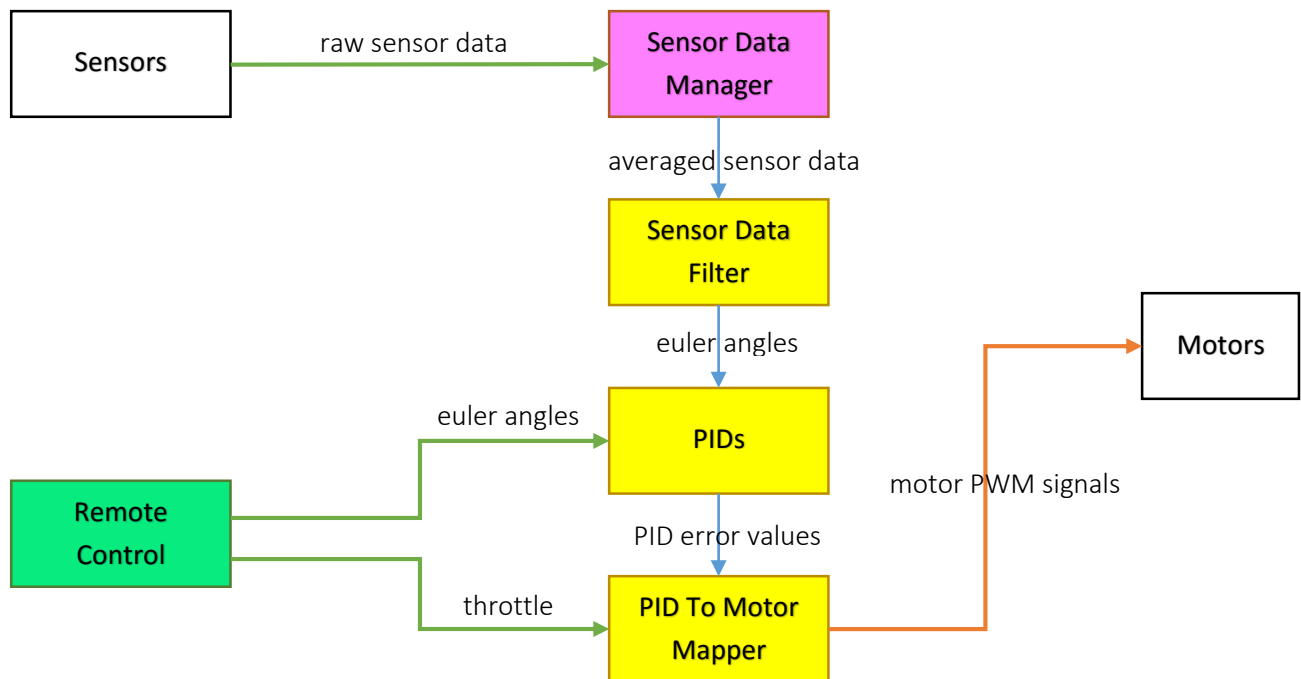


Figure 20: Dataflow through the flight controller

11.1.5 Programm Flow of the Flight Controller

An overview of the program flow and algorithm of the flight controller can be seen in Figure 21: Program flow of the flight controller and is described in the following. More details on the implementation can be found in the “Implementation of the Flight Controller” see section 11.

When the XCopter is turned on, the program starts with an initialization progress. At first the hardware device drivers, such as I²C and UART for communication, the gyroscope, accelerometer and compass drivers, the motor drivers and the PWM driver, get initialized. After that, mutexes and semaphores to synchronize the tasks and protect the critical data sections are set up. Next, the timers and finally the tasks are created and configured. When the initialization is done, the program gets divided into three tasks. The Sensor Data Manager and RC-receiver tasks are starting one second after the Main task to warm up, so they can provide the data that is needed in the Main task.

The Sensor Data Manager task, which is colored in pink, is responsible for providing the averaged data from the gyroscope, accelerometer and compass sensors. In a loop, it reads all sensors as fast as possible. When twenty data sets were read, the raw data sets are getting summed up and divided by twenty to get averaged. When the averaging is done, the main task gets notified by setting a flag, which indicates that the new averaged data is available. After notifying, the data counter gets reset and the process starts again.

The task which handles the commands the XCopter receives from the remote control is the RC-receiver task. The main loop of this task reads the incoming SUMD frames that the remote control transmits and CRC-checks for validation. If the frame is valid, the RC values are scaled and converted into Euler angles. When this is done, the RC-receiver task notifies the Main task in the same way the Sensor Data Manager task does this – by setting a flag in the Main task.

The major part of the flight controller algorithm is located in the Main task and is also running in an endless loop. At first it is checked, whether new averaged sensor data from the Sensor Data Manager is available. If there is, the Main task gets and stores this data. After storing the averaged data, it is handed over to the sensor data filter to filter the data and compute the current pitch-, yaw-, roll-Euler-angles of the XCopter. After storing the filtered data, it is checked if there are new remote control values

available from the RC-receiver task. If there is new data available, it gets also stored. The Euler angles from the filter and the remote control are then passed to the PIDs as parameters. The pitch PID gets the pitch angles, the roll PID gets the roll angles and the yaw PID gets the yaw angles. The particular PIDs then calculate an error value for each axis. Those error values are the input parameters for the motor mapper. There, the PWM signals, to run the motors at the requested velocities, are generated with the PID error values. After setting the motor speeds, the loop starts again. If there is no new sensor or remote control data available, the PIDs are fed with older data from a previous iteration. Anything else is just the same, as if there was new data available. All tasks are running until the XCopter is turned off.

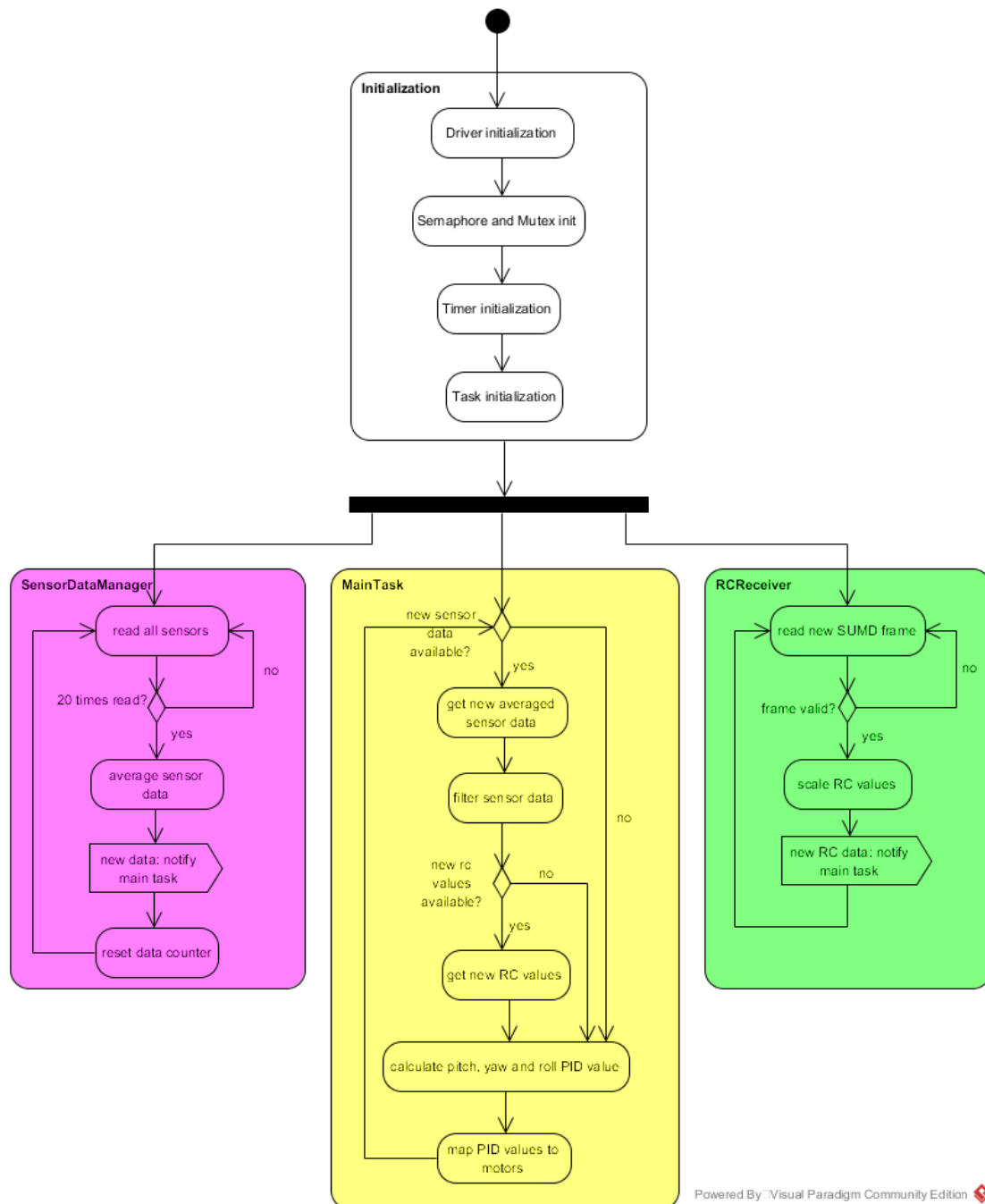


Figure 21: Program flow of the flight controller

11.1.6 Tasks and Timing

All tasks of the XCopter flight controller have to run in a specific hierarchy. The used version of the operating system $\mu\text{C}/\text{OS-II}$ has no built-in function, which is precise enough to make all tasks periodic in an appropriate way. However the Altera NIOS II HAL offers functions to create periodic alarm timers sporting a built-in timer slave IP-core. By default there is only one system timer core available which cannot be controlled by the user. This requires another timer added to the SoPC.

The currently integrated timer has a timer resolution of $1\mu\text{s}$. Thus it is possible to run all tasks in a deterministic behavior, by releasing a semaphore after a certain period of time has elapsed. The task itself has to wait for the semaphore to open.

An example implementation how tasks are made periodic can be seen below.

```
static alt_alarm periodicRCReceiverTaskAlarm;

alt_u32 mainTasktimerCallback(void* context) {
    OSSemPost(mainTaskSem);
    //must return the periode ticks
    return alt_ticks_per_second() * MAIN_TASK_PERIOD / 1000;
}

int main(void) {
    printf("Starting Program\n");
    mainTaskSem = OSSemCreate(0);
    // wait MAIN_TASK_DELAY seconds until the task starts
    alt_alarm_start(&periodicMainTaskAlarm, alt_ticks_per_second() *
MAIN_TASK_DELAY,
        mainTasktimerCallback, NULL); // periodic timer for MainTask
}
```

11.2 MOTOR PWM SIGNAL

There are three important different PWM-signals related to the XCopter project. The first is the common PWM-signal (cPWM), its range is from 0.0% to 100.0%. It is possible to send a low (0.0%) or a high signal (100%). This signal is normally produced by the XCopter. The driver provides the functionality to send every step between 0 and 255.

The second is a PWM-Signal usually used in model building (mbPWM). Usually it's running on a kHz frequency (20ms period length). The range is from 1ms high signal to 2ms high signal. It is not possible to create a constant high or low signal.

The third and maybe the most important is the one that was created for the XCopter application. It was needed because the ESC's require a non high/low signal like the mbPWM but the cPWM doesn't fulfill this requirement. It's possible to emulate the mbPWM with the cPWM, but this would decrease the resolution. For example the usable range that would be left over would be between 25 and 50. This would force the signal to stay between 1ms and 2ms. But this is not an acceptable resolution. However it is possible to calibrate the ESC's so we made them working in a range from 8 to 218. Now the resolutions is granular enough and the ESC's are working well.

11.3 INTERFACES BETWEEN COMPONENTS

In the table below (Table 4) the interfaces between the components are shown.

Notes:

* These values are not determined yet, however they do not necessarily need to be. The α - β - γ -Filter uses the input and produces the Euler angles as long as the sensor values, especially the gyro and accelerometer, are correctly calibrated.

** These values depend on the P, I and D values in the PIDs. Since these weren't finally set yet, the value ranges aren't defined. For more information see chapter 12.5.

Component			Input Range		Output Range	
			Min Value	Max Value	Min Value	Max Value
Sensor Data Manager			-	-	*	*
RC Receiver	Roll & Yaw		-	-	-180	+180
	Pitch		-	-	-90	+90
	Throttle		-	-	0	100
α - β - γ -Filter			*	*	-180	+180
PIDs	Pitch	Filter	-180	+180	**	**
		Rc	-90	+90		
	Roll & Yaw	Filter	-180	+180	**	**
		Rc	-180	+180		
MotorMapper			**	**	0	100

Table 4: Input and output ranges of the components

11.4 RC-RECEIVER

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 UART protocol, similar to a Spektrum Satellite receiver.

11.4.1 Graupner HoTT-SUMD-Signal Definition

HoTT SUMD is implemented with as a serial connection. 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 50 μ s between transmitted data bytes. The serial connection has to be setup with the following configuration:

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

11.4.2 Structure of a HoTT-SUMD Frame

A single SUMD data frame consists 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.

11.4.3 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	

Table 5 SUMD header bytes

11.4.4 SUMD Data Section Description

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

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

11.4.5 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

11.4.6 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 900 μ s length
Low position (-100%)	0x2260	Equivalent to 1100 μ s length
Neutral position (0%)	0x2ee0	Equivalent to 1500 μ s length
High position (100%)	0x3b60	Equivalent to 1900 μ s length
Extended high position (150%)	0x41a0	Equivalent to 2100 μ s length

Table 6 RC stick positions

11.4.7 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:

11.4.8 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 size. This can be calculated with:

SUMD-frame length = SUMD header length + number of channels * 2 + CRC length

11.4.9 Interpreting the Received SUMD-Frame

According to the SUMD format description, every byte has its own specific purpose. The currently received RC-commands are sliced into a high byte and a low byte, thus it is necessary to append both bytes to a 16 bit integer. Every channel value can be stored in an array.

11.5 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 into a source file "b_uartriver.c" and a header file "b_uartriver.h".

11.5.1 SUMD-Frame-High

```
VendorID: 168
Status: 1
Number of Channels in Frame: 8
Channel 0: 15200
Channel 1: 15200
Channel 2: 15200
Channel 3: 15200
Channel 4: 11872
Channel 5: 8800
Channel 6: 12128
Channel 7: 12000
CRC16 High Byte: ff00
CRC16 Low Byte: ff00
```

Figure 22 SUMD-Frame-high

11.5.2 SUMD-Frame-Low

```
VendorID: 168
Status: 1
Number of Channels in Frame: 8
Channel 0: 8800
Channel 1: 8800
Channel 2: 8800
Channel 3: 8800
Channel 4: 11872
Channel 5: 8800
Channel 6: 12128
Channel 7: 12000
CRC16 High Byte: ff00
CRC16 Low Byte: ff00
```

Figure 23 SUMD-Frame-low

11.6 SENSOR DATA MANAGER

The Sensor Data Manager (SDM) is the abstract interface for the sensors. It provides a getter method to get the sensor data. A single sensor value isn't meaningful because it is heavily influenced by the vibration jitter. Therefore the values which are provided by the getter method are simply pre-filtered with averaging. It's averaging 20 values that have been measured with high frequency. Furthermore the SDM measures the time between the current and the last value. The SDM is running in a separate task, this makes it easy to encapsulate all the functionality of the sensors in one component.

In the Sensor Data Manager is a two dimensional array (9 by 20) where the 9 is fix and the 20 can be adjusted with the VALUE_NUM define. This array is used to store all the read sensor values. Each of the 3 sensors provide 3 values, one for each axis (x, y, z). So the array is providing space for 20 (=VALUE_NUM) samples. Additionally there is an array avgData in which the last averaged data is stored. Furthermore there is a global flag called SDM_NEW_DATA_AVAILABLE which indicates that new

data is stored in the avgData-array. The avgData-array and the SDM_NEW_DATA_AVAILABLE-flag have to be accessed within a semaphore because those are the interfaces between the Sensor Data Manager task and the Main controlling task. The gyro provides a temperature value which could in theory be used to compensate the changing value offset during warm-up, but this is not implemented yet.

The way to go to access the sensor data is to check the SDM_NEW_DATA_AVAILABLE-flag and if it is set, call the getSensorData(). These functions have two parameters. Both are pointer for write back. The first pointer should point on an array of size 9 for the sensor data. The second pointer should point on an integer for the deltaT.

11.7 SENSOR DATA FILTER

The Sensor Data Filter (SDM) is responsible for filtering the sensor data and computing the current orientation of the XCopter in Euler angles. Because those calculations require a deep mathematical understanding, the sensor data filter was extracted from a third party piece of software ([1], [2]). It was configured so that it can be used in the XCopter flight controller. The implementation and mathematical details of this component are intentionally not discussed here. Because in this project the filter can be considered as a black box, only the interface to the filter is documented in the following.

The only important thing inside this black box are the parameters for the accelerometer calibration. A tutorial of how the calibration is done, can be read in chapter 15.2.

To get access to the functionality of the sensor data filter, the module provides the function filterSensorData(). This function takes three parameters:

- A int16_t pointer on the array of the averaged sensor data from the Sensor Data Manager. This is considered as an input parameter.
- A float pointer on the array where the result (filtered data) is saved. This is considered as an output parameter.
- A uint32_t number, that holds the time delta between one set of averaged data and the next set of averaged data in milliseconds. This is considered as an input parameter.

In the Main task the function gets called, whenever new averaged sensor data is available. More details of the program flow can be seen in chapter 11.1.5.

1. \XCopter\code\Utility\filterungHerrSteiper\Duocopter_PAP_Report.pdf
2. \XCopter\code\Utility\filterungHerrSteiper\FreeIMU_raw_extendef_6_ethernet_udp_3_test\FreeIMU_raw_extendef_6_ethernet_udp_3_test.ino

11.8 PID REGULATORS

11.8.1 General

To guarantee a stable hovering, UAVs often employ PID regulators and so do we with the XCopter. 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 to-be-regulated-part (in our case one of the motors), which should reach the real value the motor has currently. These two values are subtracted and the resulting difference is known as the error. 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 is set by hand (trial and error necessary). In the following the three parts of the PID regulator will be explained.

$$\text{error} = e = |\text{setvalue} - \text{realvalue}| = |sV - rV|$$

$$PID(t) = K_p * e + K_i * \int e dt + K_d * \left(\frac{de}{dt} \right)$$

Figure 24 PID-Formula

The Proportional Part

This is not a real regulator for that matter. It is 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 an example is made:

A motor should run at 100 RPM. When the P-Regulator is started and the motor is not running the error will be

$$e = 100 [RPM] - 0 [RPM] = 100 \text{ and it will return } K_p * e \text{ to the motor.}$$

The Integral Part

This is slowest part of a PID because it is 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 sensitive part of the PID, because it reacts to temporal changes. So the faster the real value (or sensor data) changes, the higher the return value. This part does not depend on the RC signals. This is useful for fast response which is important in the system because it has to be stabilized at all the time.

It is assumed that the XCopter is hovering stable and a wind gust is hitting the XCopter, then the P-Part will recognize and steer against it in proportion to the error. But this too 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.

11.8.2 PID Regulators in the Simple Flight Controller

Structure of the PID Components

One of the most important software parts of the flight controller are the PID modules. These modules are responsible for the orientation of the XCopter and guarantee a stable movement. There are three PID modules implemented in the flight controller. Each axis (roll, pitch and yaw) is controlled by one PID regulator. As described in section 11.7 every PID module gets two input parameters, on the one hand the set point parameter and on the other hand the real value. The outputs of the RC receiver module are providing the set points, and the returned values of the sensor data filter are taken for the real values of the PIDs (0).

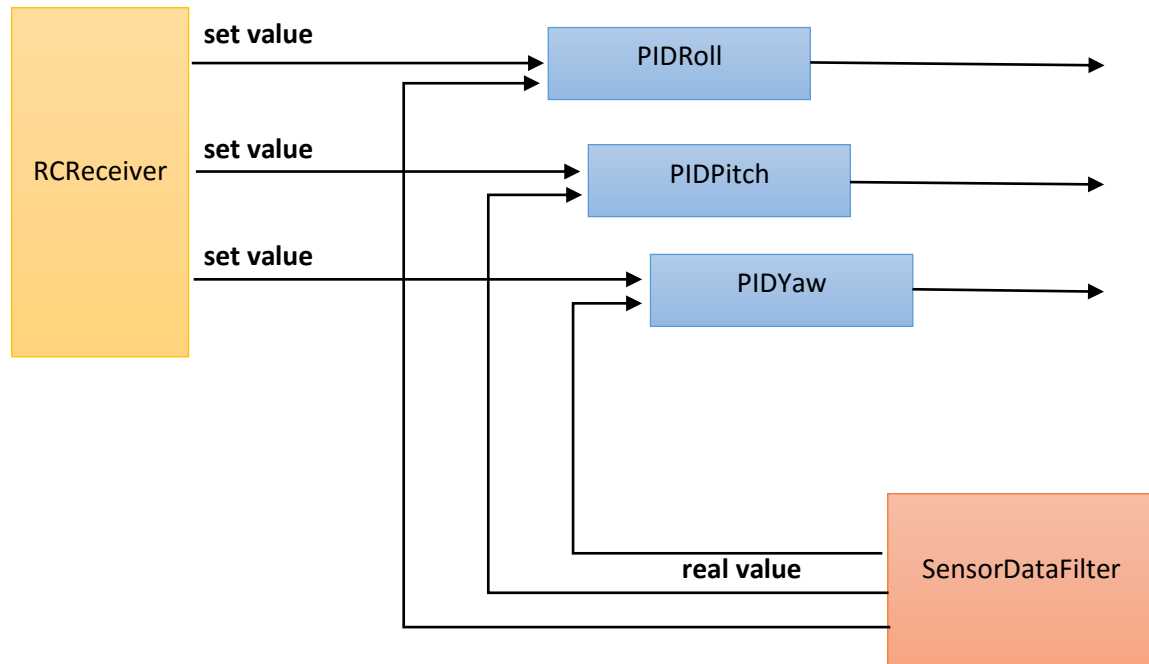


Figure 25 Structure of the PID components

The PID modules provide a function which expects the parameters set point and real value and returns a float number which represents the control variable for the motors. In the function the proportional part, the integral part and the differential part are computed as described in section 11.7.

The constants K_p , K_i , K_d (in this function named RollKi, PitchKi, etc.) regulate the behavior of the PID controllers. Incorrect values of the constants can lead to a crash of the XCopter. Therefore, the constants have to be set absolutely correct. These values (RollKp, RollKi, RollKd, PitchKp, PitchKi, PitchKd, YawKp, YawKi, YawKd) need to be determined experimentally, using for example a specific tuning method like the "Ziegler-Nichols-Method" or using an online tutorial for tuning PIDs of a quadcopter ([12]). Additionally, a limitation of the maximum and minimum return value is implemented, so the value is also valid in case of a wrong computation. The maximum and minimum values can also be changed while experimenting with the constants of the PID regulators.

11.9 PID TO MOTOR MAPPER

11.9.1 The Idea Behind the PID to Motor Mapper Module

After the three PID regulators computed the values for the next orientation of the XCopter, it is necessary to take the three output values as well as the throttle value of the RC receiver module for calculating the PWM signal for each motor. This is the task of the PID to motor mapper module. The motor mapper module was developed to solve two problems of mapping the input values to each motor. The first problem is, that a mathematical formula is needed which gets three PID values as well as the throttle value and computes a correct PWM signal for every motor. This formula, also named mixing table or mapping table, already exists in other quadcopter projects and could be adopted. The second issue is to write a function which provides the boost which is needed to balance out the Xcopter (the boost that is generated by the PID values) and is not bigger than the boost that keeps the Xcopter in the air (the boost which is generated by the throttle value). Additionally, this function must ensure that the sum of the “throttle boost” and the “PID boost” do not go over 100%.

11.9.2 Explanation: the Mapping Table

The following code extract shows the mapping table of the PID to motor mapper module. The function PIDMIX gets a throttle value from the RC receiver and the roll, pitch and yaw value from the PID modules. The real mapping table are the constants with the plus ones and minus ones. The function multiplies the roll, pitch and yaw value with the constants of the mapping table and sums it up with the throttle value. The return value represents the PWM signal for the motor.

```
//Mapping table for a QUADX configuration
motorQuadx[0] = PIDMIX(-1,+1,-1, throttle,roll, pitch, yaw); //REAR_R
motorQuadx[1] = PIDMIX(-1,-1,+1, throttle,roll, pitch, yaw); //FRONT_R
motorQuadx[2] = PIDMIX(+1,+1,+1, throttle,roll, pitch, yaw); //REAR_L
motorQuadx[3] = PIDMIX(+1,-1,-1, throttle,roll, pitch, yaw); //FRONT_L
```

11.9.3 Explanation: Ensuring the Correct Boost of the PIDs

The second problem which is described in section 11.9.1 can be solved by limiting the throttle value to 75% and using a formula, which is described below, to compute the relation between the throttle boost and the PID boost. The advantage of limiting the throttle value to 75% is that there is a scope of +25%/-25% for the PID boost. A problem that can appear now, is that the PID boost could get a bigger value as the value of the throttle boost. In this situation the model will not go up anymore. The following formula is used in a while loop in the motor mapper module to decrease the value of PID_{mix} and to adjust the value of *throttle*.

$$C_{th} * throttle + C_{mix} * PID_{mix} = 100\%$$

C_{th} and C_{mix} are the constants which were used to adjust the PID_{mix} and the *throttle*. The two constants are both initialized with the value one and in case that the value of PID_{mix} is greater than the value of *throttle*, C_{mix} will be decremented by 0.1 and C_{th} will be adjusted with the formula:

$$C_{th} = \frac{100\%}{throttle + (C_{mix} * PID_{mix})}$$

The following code snippet computes the values for the *throttle* (throttleMix) and PID_{mix} (pidMix), checks if pidMix is bigger than the throttle value called limitedThrottle and returns the sum of

throttleMix and pidMix, which represents the value for the PWM signal. In case that pidMix is bigger than limitedThrottle, the function decrements CMix and computes the values again.

```
while(CMixIsCorrect == 0) {
    //Compute the values using the mapping table
    pidMix = CMix * ((roll * X) + (pitch * Y) + (yaw * Z));
    throttleMix = limitedThrottle * computeCThrottle(limitedThrottle, CMix,
pidMix);

    if(pidMix > limitedThrottle) {
        CMix -= CMix > 0.1 ? 0.1 : 0;
    } else {
        CMixIsCorrect = 1;
    }
}
return throttleMix + pidMix;
```

The function computeCThrottle() computes the value for the constant C_{th} .

```
float computeCThrottle(float limitedThrottle, float CMix, float pidMix){

    //Compute the CThrottle value subject to CMix
    float CThrottle = (float) (100.0 / (limitedThrottle + CMix * idMix));
    return CThrottle;
}
```

After calculating the PWM signals for the motors the function writeToMotors() is called, which sets the motor speed for each to one. The motor speed is indicated in percent. The mapper module is not validated at the moment. Because the constants for the PIDs are not available yet. More information about that can be read at section 12.5.

11.10 SPARKFUN SENSOR BOARD

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 for exact information [1] [13]. 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. The accelerometer and the gyroscope must be calibrated [13] [14] [15] [1].

11.10.1 Calibration of the 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. These 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 in chapter 15.2 describes the procedure to get the values. The output data of the accelerometer is shown in Figure 26. The offset value is the result of the difference between zero and the measured value zero point.

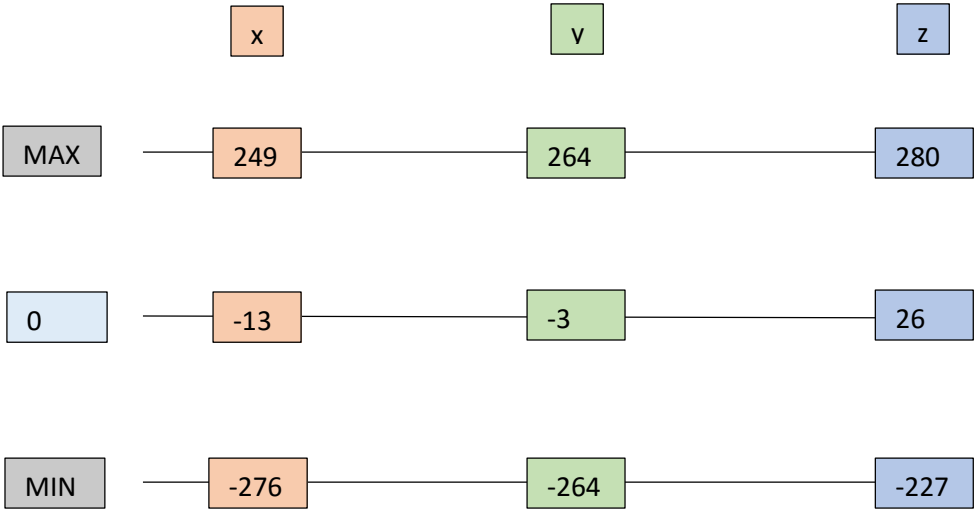


Figure 26: Output data of the accelerometer

The chart in Figure 27 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.

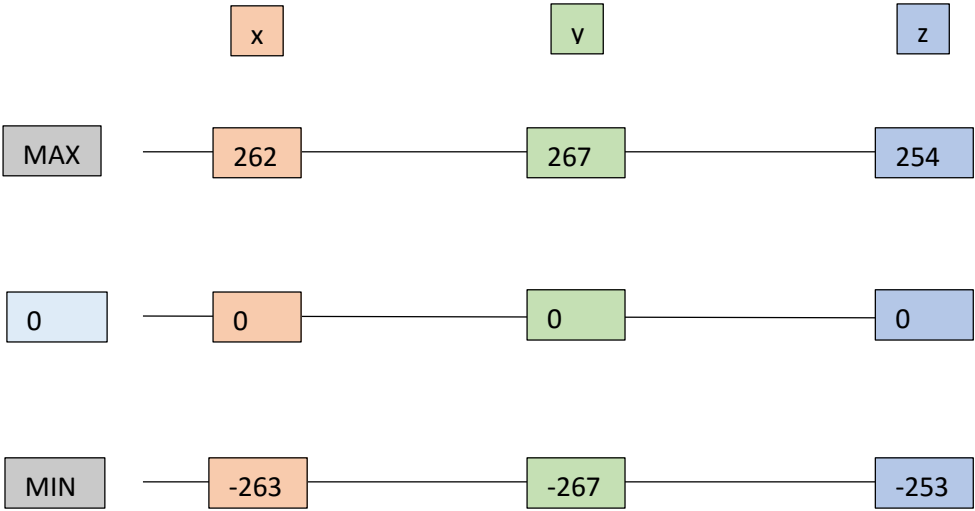


Figure 27: Output data minus offset value

11.10.2 Calibration of Gyroscope

A gyro has no defined state after being started and there is an error offset, especially if the sensor has been moved while it was switched off. Additionally the sensor has a warmup phase in which 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 gyroscope the Sensor Data Manager provides an init-method which takes samples and compares them until the difference between the samples is small enough. After that it saves the last sample as offset and the gyro meter is initialized.

11.11 LOGGING

Logging is an important part, especially for future debugging purposes. The target of the logging on the NIOS II system is to provide all useful data from the sensors to the developer, additionally it gives the opportunity to display information on the ground station while flying. One of the goals is to be able to log raw and filtered data.

As a task on the system, the logging has a low priority because it should not use calculation power, which actually would be needed by the much more important flight controller. The logging task is running as a filler in the time slots between the more important tasks.

The data, which needs to be logged is produced in the main controlling loop. The end of such a loop is the perfect place to start the logging procedure. Logging needs to be encapsulated from the control loop. In the flight controller loop all data which should be logged, is pushed into a queue. If there is computing time left (no task is running) the logger task starts and is reading data out of the queue and sends it to the ARM/Linux system via MCAPI.

12 CHALLENGES AND PROBLEMS

12.1 GENERAL PROBLEMS AND CHALLENGES WITH THE FLIGHT CONTROLLER

During the planning and implementation of the XCopter flight controller, the team encountered several problems that interfered the development of the flight controller. Most of those problems were the lack of experience and expertise.

The most general problem was, that most of the team members never dealt with UAVs or RC model making before. Therefore nobody really knew what a flight controller is or how a flight controller works. This know-how is essentially important for developing an own flight controller. The problem was more or less solved by reading articles, tutorials and documentations of open-source solutions. Because the XCopter flight controller is implemented on an Altera DE1-SoC Board and the sensors and motors are not standardized, the code of most of the free and open-source solutions didn't help a lot.

Another problem was that no team member knew, how to design and structure a software system with a medium to high complexity like the flight controller. Because of this, the designing phase took an above average time to finish. It was needed to change the design several times, which of course leads to regression in progress. When the design of the flight controller was done and the work was distributed to the team members, there was an additional problem with specifying the interfaces of the single components.

The interfaces were an issue because of the missing expertise in sensor-, controlling- and regulation-technology. To connect the particular components of the flight controller, it is essential to know how the data has to be interpreted and converted in the respective parts. One major challenge was to convert all the sensor and remote control data, so that the ranges and units fits together. When the ranges or units don't fit together, the PID regulator as well as the motor mapping cannot work properly. Especially when configuring the PIDs and the motor mapper, know-how in regulation-technology would have helped a lot. Because the error of distributing work without bringing out the interfaces was committed, the problem with the data ranges and units was realized at a very late state. This isn't solved in the current version of the flight controller.

Because the development takes place on the Altera DE1-SoC, a special version of the Eclipse IDE has to be used. In this special version, when a program for the DE1-SoC board has to be developed, the IDE creates two projects. One project is where the own source code and files are added and the other project is a so-called BSP-project. The acronym BSP stands for board support package. This project is generated with the SoPC information file, which contains all the hardware addresses of the devices on the board. That BSP-Project is then linked to the actual (own) project to make the devices available there. To develop professional software, a version tracking system such as Git or SVN is mandatory, so that the team members can share and merge their work easily.

Normally one would just create a new Git repository containing the software project and every team member would push and pull their work to or from the repository. The laboratory computers, which were used, had no pre-installed Git system so it had to be installed on them later on. The laboratory PCs are multi-user workstations and the Git software that was used didn't work well. Whenever another user logs in to the PC, GIT gets reset and the repository has to be cloned again. Because of the special environment in the lab and the special IDE, Git is not working properly, and the code cannot be built and tested all computers. This led us to use one computer as the main developing machine, where the code is built and loaded onto the DE1-SoC board. This of course aggravated the workflow of the team and in the whole project.

12.2 PROJECT-MANAGEMENT

12.2.1 Time- and Team-Management

At first our team consisted of nine people which was reduced to eight during working on the project. Due to the nature of our curriculum we weren't able to effectively coordinate dedicated times to work on the project. This was especially a problem in the fifth semester (second half of working on the project), where every student had to choose three elective courses which took place at different times. It was almost impossible to find a fixed weekly date where everyone had time to work on the project as a complete group, especially with a team with so many members. This made synchronization of knowledge about the state of the tasks being worked at amongst team members a problem.

12.2.2 Scrum

Scrum is a very good management system for software development. Development with Scrum is an agile, iterative and incremental procedure. Sadly, Scrum didn't yield the productivity that our team hoped for. Scrum is normally meant to be used by small teams that are working together closely on a daily basis to reach a common goal. Our curriculum however does not allow for daily work on the project. We were only able to dedicate two or three days a week to the project, which rendered Scrum less effective than it could have been. Scrum meetings took up a lot of time with our group of eight to nine individuals. It was hard to stay focussed in the meetings with so many people as discussions often drifted into regions that were hardly relevant for the tasks at hand. The more people a group consists of, the more thoughts go into a discussion and sometimes it takes too much time to evaluate each and every opinion which generates a time-consuming overhead.

Sprint planning meetings at the beginning of each sprint were held to get an overview about what should be the output of each sprint. Each sprint planning meeting took up several hours of time for the whole team that could have been dedicated to actively working on the project. Creating user stories and tasks took up too much time. This was because in sprint planning meetings the whole sprint, which was about six weeks of work in our case, had to be planned in meticulous detail. Task-priorities were determined in a democratic fashion with votes from every team member. These sprint planning meetings led to further discussions which took up even more time than the daily Scrums. Although Scrum is meant to reduce time spent on planning the procedure of development, in our case it had the opposite effect and increased overhead.

12.2.3 JIRA

We used a project management tool called JIRA for managing our tasks and keep track of priorities and dependencies. After each sprint planning meeting, the Scrum-Master had to update JIRA with new tasks and priorities for the sprint. The upkeep of JIRA took away a lot of time for the Scrum-Master which could have been better spent in actively working on the project, especially because most team members didn't actively use JIRA. Not using JIRA wasn't because of laziness though. The team was well aware about what had to be done, even without JIRA. JIRA was simply an inconvenience for our project and was dropped later on.

12.2.4 Git

It was decided to use Git as the management system for resources like documents, datasheets and source code of our project. Further it was decided to use SourceTree from Atlassian as a Git-client. SourceTree has an easy to use graphical user interface and has a faster learning curve than Git's terminal-like interface. Although easy to use, some team-members had problems getting SourceTree to synchronize their files reliably on their personal computers. Git also behaved in a weird way when installed the computers in the laboratory (chapter 12.1). Another problem was incompatibility with the Eclipse IDE. Sometimes team-members simply forgot to pull to have the most recent resources which

could have been circumvented with a feature that pulls from the repository automatically every five minutes.

12.3 LINUX AND BUILDROOT

We were a bit late in choosing members that specialized exclusively in understanding the Embedded Linux system and the VM that is used to cross compile code for ARM. It sometimes happened that one of our Linux specialists was not in the laboratory when a quick update of code should have been tested on the embedded OS which led the development to a grinding halt at times. This problem stemmed from the fact that our curriculum was so diverse in the fifth semester.

The team underestimated the time it would take to understand the system and get enough experience to work with it reliably. Many stumbling stones occurred during development like getting stable WiFi working or realizing that the date had to be set before using secure copy (SCP) to the ARM-system.

At first we thought that we could implement a Wi-Fi-driver for Embedded Linux on ARM ourselves. After two weeks of research and work we scrambled this idea as it took up too much resources. We had to look into what integrated drivers Buildroot offered and chose a Wi-Fi-dongle with a supported chip accordingly. It has to be stated that even if Buildroot offers a driver it still doesn't mean that everything works out of the box if implemented into the kernel. Configuration of the Wi-Fi-stick (especially the WPA-suppllicant part) took up a considerable amount of time.

Working with Buildroot wasn't easy. Although it is meant to be easier to use than similar tools for building custom Linux distributions, it has still a steep learning curve. The design of the Buildroot user-interface is not very self-explanatory and sometimes check boxes for customization options are hidden behind dependencies. We weren't able to find a driver for our Wi-Fi-stick before checking some, at first seemingly unrelated, options in Buildroot. The steps to bring an updated image of the operating system onto an SD-card have to be followed precisely and it was hard get everything right when not working with Buildroot on a daily basis. Things that were clear three weeks in the past had to be re-learned at times because of the unintuitive nature of the system. The tutorials from Mr. Strahnen in the Docs folder of the VM were a great help for working with Buildroot though.

12.4 USB-CONTROLLER AND 3D-MAPPING

To get the multi-copter to the point where it is able to map its environment in 3D is a very hard task. A lot of time was spent finding a suitable USB-controller that matches all the requirements (pin count, linux drivers, data transfer rate) of our system. At one time a suitable controller was found but it was only available for the automotive sector so availability on the free market had also be taken into consideration.

But even after finding a suitable USB-controller, problems still bubbled up. The raw computing power needed for 3D-mapping from two Kinects in real time is very high. Mr. Steiper stated that on his private multicopter, a dedicated mobile Intel i7-CPU is doing this task solely and it is still only able to process the image at a low framerate. The assumption that the ARM-CPU on the DE1-SoC-Board would suffice for this task had to be corrected. New 3D-cameras that need less performance have to be found. But even if performance wasn't a problem, finding Linux drivers for our ARM-processor could pose quite a challenge as 3D-cameras haven't penetrated mainstream in a big way at this point in time and custom drivers for ARM are rare. Own drivers would have to be written if no Linux drivers are to be found which could take up a whole two-semester project for a group of students (if even possible). In the end, the Task of getting 3D-mapping to work had to be dropped as it would have taken up most of the time left for the project. The problems that showed up while working on the USB-controller are documented in the corresponding chapter in more detail (chapter 9.4).

12.5 PID AND MOTOR MAPPER

At this point in time the PID and the motor mapper modules (PIDToMotorMapper.c) are implemented with some issues. The PWM output values of the motor mapper module aren't correct. A big source of errors could be the wrong input values, that are influenced by a wrong conversion of the different ranges between the flight controller modules. But it also cannot be put out of the question that there is a logical error in the motor mapper module itself. Another challenge is that the output ranges of the PID modules are changing, if the constants of the PIDs are modified. So the input values of the motor mapper module must be adjusted every time the PID constants are altered.

13 LESSONS LEARNED

13.1 PROJECT MANAGEMENT

- A team of eight to nine students is too big to effectively work on one project. Even if the team is divided into smaller groups, communication suffers from the size of the whole team.
- Reducing overhead by choosing the right method to manage the project is of high importance. Scrum was not the right choice for our project. It is probably way better suited for teams that are working on projects on a daily basis. Managing user stories and tasks takes up too much time when the group is only able to work actively on the project on two to three days a week. But even with the drawbacks of Scrum for the XCopter project, there was an undeniable learning effect in having to work with such a system.

13.2 LINUX AND BUILDROOT

- It is necessary to choose someone, or even better a small group, who specializes in working with Linux.
- It is essential to get to know Buildroot when adding drivers to the Kernel. The Buildroot webpage has a great manual that ([16]) describes everything in detail. The Documents from MR. Strahnen located on the VM are also a great help.
- Some basic knowledge about how makefiles and (cross-)compilation in general works should be acquired before developing for the ARM-system.

13.3 USB-CONTROLLER AND 3D-MAPPING

- This task will take up a huge amount of time. A group of five people working exclusively on this problem for one or two semesters could be sufficient to get this feature to work.
- Kinect cameras are not suitable for the DE1-SoC system. The demanded computation power for two Kinects is simply not available.
- Alternatives for Kinect will have to be chosen. One major point in considering 3D-cameras is the computing power that is needed to handle 3D-image data.
- It is no easy feat to choose the right USB-controller that is compatible with the DE1-SoC Board and checks all the boxes on the list of features that are needed to use USB driven 3D-cameras.

14 FUTURE WORK

The development of the flight control unit has stopped at a late state. Future steps encompass the connection of all major control loop modules (e.g. PID and Motor Mapper module). This can be implemented immediately on top of the current state of the flight control unit. Furthermore the timings of all control loop tasks have to be evaluated. After all parts were put together the system can be tested in a controlled environment like some sort of test construction.

Another important part of a proper and user friendly flight controller is to offer some health status data. In ideal conditions it should be possible to monitor all available data i.e. sensor-, RC-receiver-data as well as the remaining battery life. However the hardware and all functions to measure battery life have to be implemented first. It was planned to be logged and transferred to a ground station since the initial planning phase. The current development state already covers the most fundamental parts. It is possible to connect the flight control unit to a ground station software via a so called MAVLink protocol. However transferring the logging data still has to be fully implemented to the flight control program. Example programs showing how to transfer data to the Linux HPS system are available. This simply has to be placed in the main flight control program. The section where the logging data has to be transferred during the runtime will be signed in the program code. However both parts only were tested separately. Since the used RC receiver is capable of handling telemetry data. It would be feasible to transmit some parts to the RC transmitter via the transmitting serial connection to the receiver.

In the far future the XCopter is meant to be capable of flying autonomously. The main collision detection shall be controlled by the second NIOS II CPU. In order to accomplish such functionality, some additional sensors must be available e.g. ultrasonic sensors.

15 HOW-TO SECTION

15.1 ARM/LINUX: VIRTUAL MACHINE, CROSS COMPILATION, EXECUTING CODE

15.1.1 Virtual Machine Usage and Folder Structure

The most recent version of the virtual machine, that was used to cross compile code for the embedded Linux system is located in:

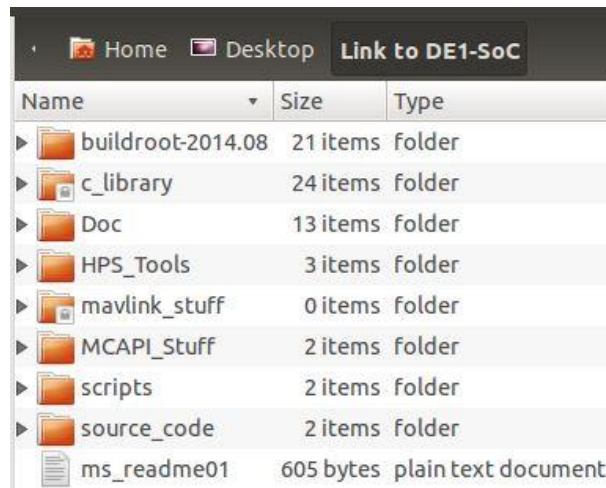
`C:\USERS\Public\QuadCopter\Ubuntu_1204_64bit_de1soc_v03`

on the CTLAB-08 computer. The VM can be opened with the VMware Player that is already installed on most of the computers in the lab. The VM itself is running a 64bit Ubuntu 12 distribution. The system can be accessed with the **username: de1soc** and the **password: hsu**. This user is not running as root natively, so keep in mind that for some actions administrative rights may have to be added (sudo).

All the relevant files are located in the folder:

`/home/de1soc/DE1-SoC`

This folder can also be accessed through a link that is placed on the desktop. It contains the following subfolders:



Name	Size	Type
buildroot-2014.08	21 items	folder
c_library	24 items	folder
Doc	13 items	folder
HPS_Tools	3 items	folder
mavlink_stuff	0 items	folder
MCAPI_Stuff	2 items	folder
scripts	2 items	folder
source_code	2 items	folder
ms_readme01	605 bytes	plain text document

Figure 28: Folder structure of the VM

- **Buildroot-2014.08**
 - Files for execution and configuration of Buildroot, changing the Linux Image, boot loader image and root file system image.
- **C_library**
 - Test files (heartbeat, hello_arm...) and libraries for MCAPI
- **Doc**
 - Essential documentation files from Mr. Strahnen. **This is the most important folder. Reading these documents is highly recommended!**
- **HPS_Tools**
 - Files for successfully booting up the DE1-SOC System (not interesting for development)
- **Mavlink_stuff**
 - C-Code for testing MAVLINK communication (further information at qgroundcontrol.org/dev/mavlink_linux_integration_tutorial).

- **MCAPI_Stuff**
 - Everything that has to do with MCAPI communication.
 - It is highly recommended to read *howToStartMCAPI01.txt* in this folder
 - *MCAPI_SYS* folder is for deeper understanding of the protocol, but not necessary for development.
 - *MCAPI_Apps* contain a lot of test programs to test MCAPI functionality. A good first test would be to try to get *MCAPI_packetTest_receivingNode_cpuS0* (running on NIOS!) and *MCAPI_packetTest_sendingNode_cpuM* (running on ARM/Linux). This test exchanges some data with different package sizes. Do not get confused by the naming of the test programs. Both programs receive and send data! **The most important thing** is to run the receiving program on NIOS and the sending program on ARM/Linux.
 - The latest communication test is in the folder *XCopter_files* and contains example code for sending a 512-byte package.
- **Scripts**
 - Contains two deprecated scripts. Feel free to insert useful scripts here.
- **Source_code**
 - *Network_config* contains three files that describe how the network is set up on ARM/Linux. These files can be taken as reference if wlan problems should occur.
 - *Test_files* is an empty folder for temporary code

The cross compiler is only set up on this Ubuntu Virtual machine. You can copy the VM to other PCs if needed. As no cross compiler is set up for a windows system, it is recommended to use the VM exclusively for development for the ARM part of the DE1-SoC platform. At a first glance using a cross compiler seems a little unwieldy and complicated, but there are two main reasons to use a cross compiler in this scenario. Cross compilers are used widely by professionals and it is good to have seen how it works. The ARM CPU is not as fast as an Intel machine running the VM. Compiling directly on ARM would take a lot more time. Also development in an embedded Linux terminal is way harder than in Ubuntu.

For development of NIOS II applications windows is the first choice, as altera provides a relatively solid development platform based on Eclipse. There is a HowTo PDF-File from Group Bumblebee (2014) that goes into detail about how to develop for NIOS II with the tools provided by Altera on the DVD with the resources for this project.

15.1.2 Cross Compiling Code on Ubuntu for ARM

To be able to compile code for ARM/Linux on another system (Ubuntu in this case), the correct cross compiler has to be set in the corresponding makefile. The cross compiler is located in:

```
/home/de1soc/DE1-SoC/buildroot-2014.08/output/host/usr/bin/arm-buildroot-linux-  
uclibcgnueabihf-cc
```

An example for how to compile a file with the cross compile:

```
/home/de1soc/DE1-SoC/buildroot-2014.08/output/host/usr/bin/arm-buildroot-linux-  
uclibcgnueabihf-cc -o <filename of output> <filename of code> -I ./common/
```

You can load your executable onto the ARM-system via SCP.

15.1.3 Connect to the ARM Linux System

To connect to the Linux system it's necessary to start the Ubuntu VM and to plug in the mini-USB cable into the NIOS Board. Be sure that the system is powered on (easy to see when the red digits are on).

1. Start a new Terminal
2. Type in "sudo Kermit" to connect to the linux system
3. Type in the password for de1soc ("hsu") press enter and wait a second.
4. The connection should be establish in less than 10 seconds. It's possible that you have to press Enter before you can see the welcome screen of the ARM Linux system
5. login with the username "root" and the passwort "hsu"

If the connection is established successfully, the terminal should look like this:

```
de1soc@ubuntu: ~
de1soc@ubuntu:~$ sudo kermit
[sudo] password for de1soc:
?OpenSSL libraries do not match required version:
. C-Kermit built with OpenSSL 1.0.0e 6 Sep 2011
. Version found  OpenSSL 1.0.1 14 Mar 2012
OpenSSL versions prior to 1.0.0 must be the same.
Set LD_LIBRARY_PATH for OpenSSL 1.0.0e 6 Sep 2011.
Or rebuild C-Kermit from source on this computer to make versions agree.
C-Kermit makefile target: linux+krb5+openssl
Or if that is what you did then try to find out why
the program loader (image activator) is choosing a
different OpenSSL library than the one specified in the build.

All SSL/TLS features disabled.

Connecting to /dev/ttyUSB0, speed 57600
Escape character: Ctrl-\ (ASCII 28, FS): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
-----
Welcome to X-Copter(socfpga-3.10-ltsi-rt)
xcopter login: root
```

Figure 29: Kermit open connection

Problems with the connection

Hints: if the Terminal looks like this:

```
de1soc@ubuntu: ~
de1soc@ubuntu:~$ sudo kermit
[sudo] password for de1soc:
?OpenSSL libraries do not match required version:
. C-Kermit built with OpenSSL 1.0.0e 6 Sep 2011
. Version found  OpenSSL 1.0.1 14 Mar 2012
OpenSSL versions prior to 1.0.0 must be the same.
Set LD_LIBRARY_PATH for OpenSSL 1.0.0e 6 Sep 2011.
Or rebuild C-Kermit from source on this computer to make versions agree.
C-Kermit makefile target: linux+krb5+openssl
Or if that is what you did then try to find out why
the program loader (image activator) is choosing a
different OpenSSL library than the one specified in the build.

All SSL/TLS features disabled.

?SET SPEED has no effect without prior SET LINE
Sorry, you must SET LINE or SET HOST first
C-Kermit 9.0.302 OPEN SOURCE:, 20 Aug 2011, for Linux+SSL+KRB5 (64-bit)
Copyright (C) 1985, 2011,
Trustees of Columbia University in the City of New York.
Type ? or HELP for help.
(/home/de1soc/) C-Kermit>
```

Figure 30: Kermit connection failed

- Check all cables
- Try another USB port
- Be sure that the USB is connected to the vmware player (figure below)

- Be sure that you connect with super user privileges (“**sudo** Kermit”)
- Try to restart the VM while the NIOS board is powered on

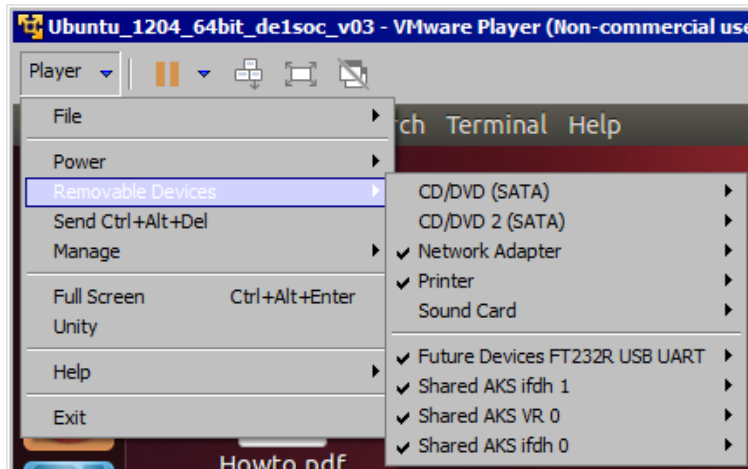


Figure 31: Connected USB Devices of the VM

15.2 GETTING CALIBRATION PARAMETERS OF THE ACCELEROMETER

For a detailed description of how to do the calibration, see chapter 11.10.1.

1. To get the correct maximum values, minimum values and the zero point a rectangular angle is required. So may be the best way is to fix the sensor board onto a rectangular form (piece of wood).
2. Open the flight controller project in the IDE and use the function “test_9dofStick()” in the file “test_9dof.c” to display the values of the sensor.
3. The orientation of the axes is imprinted on the face of the sensor board.
Because of the jitter it is much better to take an average of 300 measured values.
4. Place the rectangular form in such a way that the axis of the sensor is in the maximum position.
5. Place the rectangular form in such a way that the axis of the sensor is in the minimum position.
6. Place the rectangular form in such a way that the axis of the sensor is in the zero-point position.
7. Repeat step 4 to 6 for each axis (x, y, z).

15.3 CHARGING BATTERIES FOR THE XCOPTER

Info: This little tutorial describes how to charge our batteries for the XCopter-Project

	Identifier:	Information:
Hardware:	Battery1 (LiPo_01)	Name: FlightPower Hacker evo 20 4900mAh 6S1P Charging: continuous: 20C, burst: 50C, charging: 1C Stored in: Speicher[31] as LiPo_01
	Battery2 (LiPo_02)	Name: XTRON 40C 5000mAh 6S1P Charging: continuous: 40C, burst: 80C, charging: 4C Stored in: Speicher[20] as LiPo_02
	Battery3 (NiCd_01)	Charging: loading current: 110mAh Stored in: Speicher[19] as NiCa_01
	Charging-Station:	Ultra Duo Plus 60

Table 7: Battery charging programs

We saved three configurations in the Ultra Duo for our batteries named: Lipo_01, Lipo_02 and NiCd_01

Battery Overview		LiPo_01	LiPo_02	NiCd_01
Cell-Count:		6	6	4
Nominal	Cell-Voltage	3.7V	3.7V	
	Total-Voltage	22.2V	22.2V	
Max.	Cell-Voltage	4.2V	4.2V	
	Total-Voltage	25V	25V	
Min.	Cell-Voltage	3V	3V	
	Total-Voltage	18V	18V	
Max. Charge Current		4.9 A (1C)	20 A(4C)	

Table 8: Battery specs

Hint: The maximum charge power is not above 80 W.

Important: Warnings and security information are found in the documentantion-pdf: 6478_ULTRA DUO PLUS 60_de.pdf

You should read that before using the charging station as the information therein will not be in this tutorial.

15.3.1 Charging Batteries:

Current charged per charging session = current * charging time

Refer to the datasheet of your batteries for information about max. current for charging.

Standard-charging-current is 1/10 of the capacity (1.7Ah capacity -> 170mA standard-charging-current)

1. Plug power cable to start charging device
2. Connect charging cables to charging station (red = plus, black = minus)
3. Recommend for LiPo's: Connect cell adapter to "Balance" board and "Balance" board to charging station
4. We will use the CC/CV charging-mode for our batteries as fast-charging not supported by every type

15.3.2 Step-by-Step-Solution

Overview



Figure 32: Battery charger

Control Interface

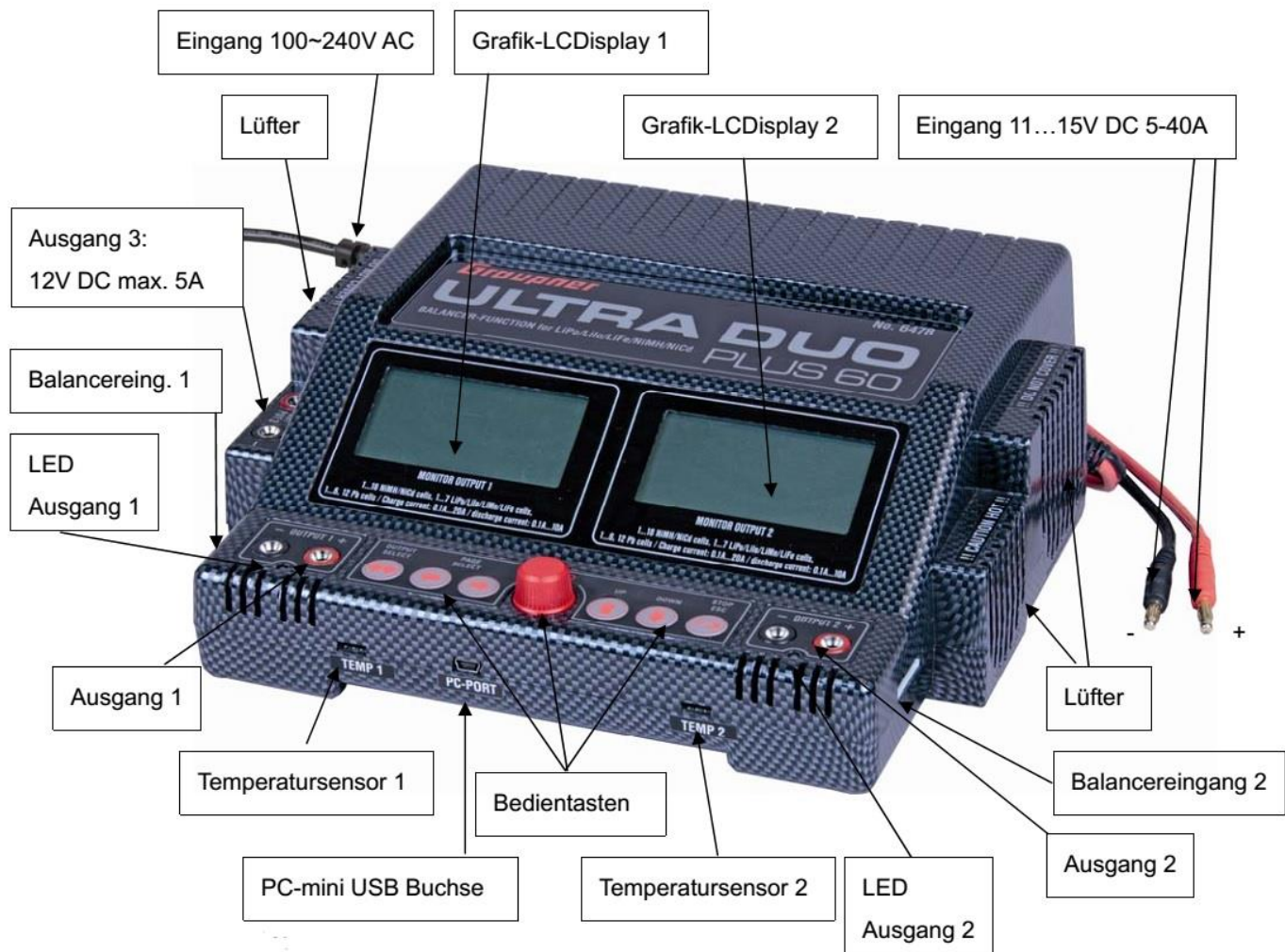


Figure 33: Battery charger overview

Enter "SPEICHER"-item: turn red click wheel (RCW) to select "SPEICHER" and push RCW

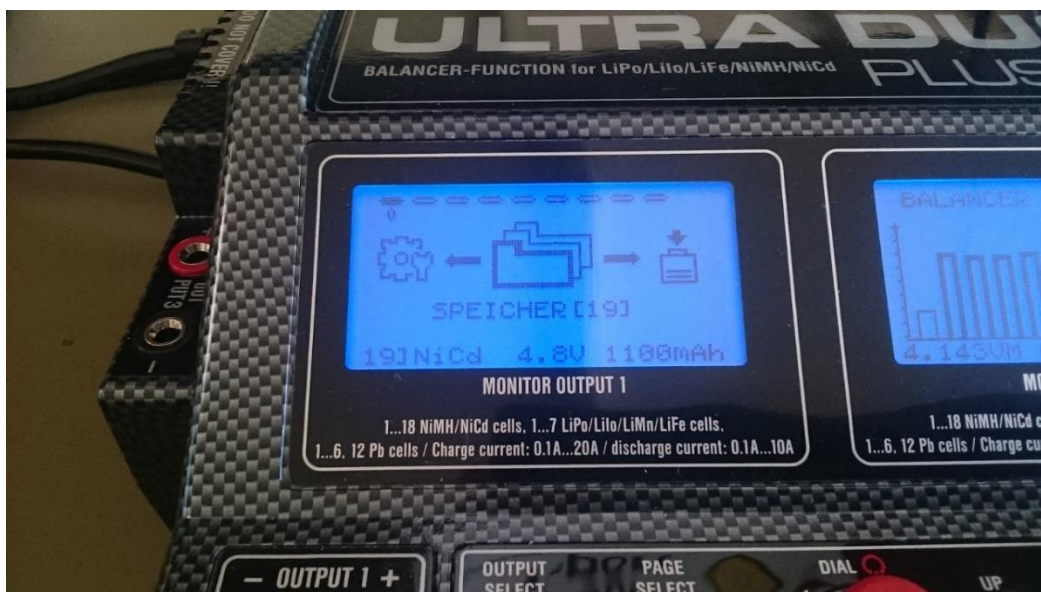


Figure 34: Battery charger menu

Choose save state with wheel and push STOP/ESC button to confirm (Hint: don't push RCW! With pushing RCW you overwrite save states!)

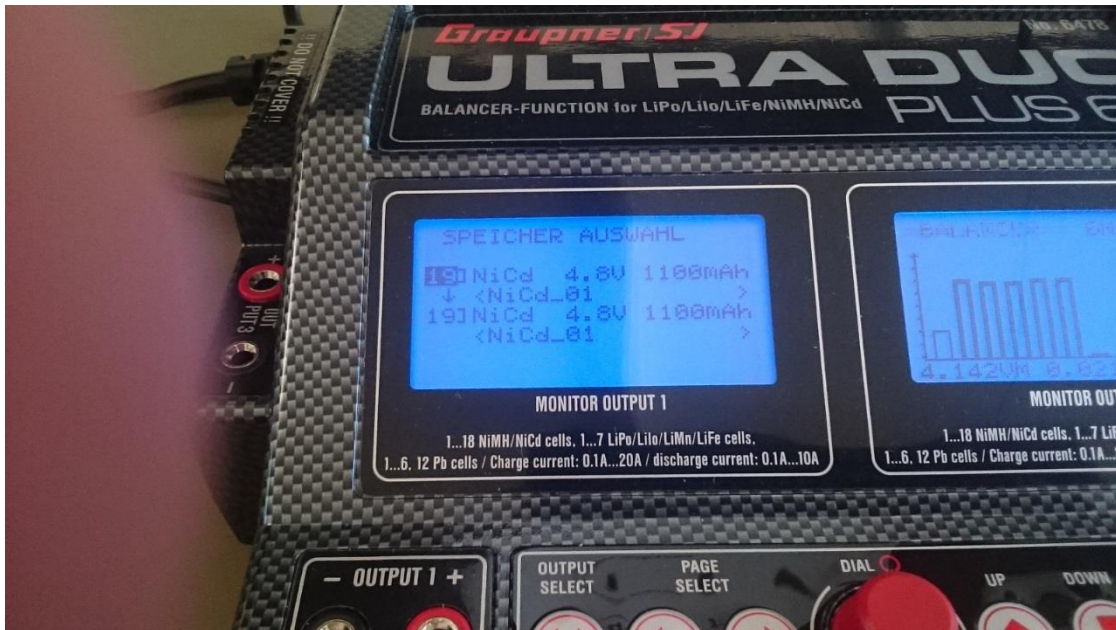


Figure 35: Battery charger select program

Choose Function: "LADEN" or "ENTLADEN" with RCW (here you can see the charging mode)

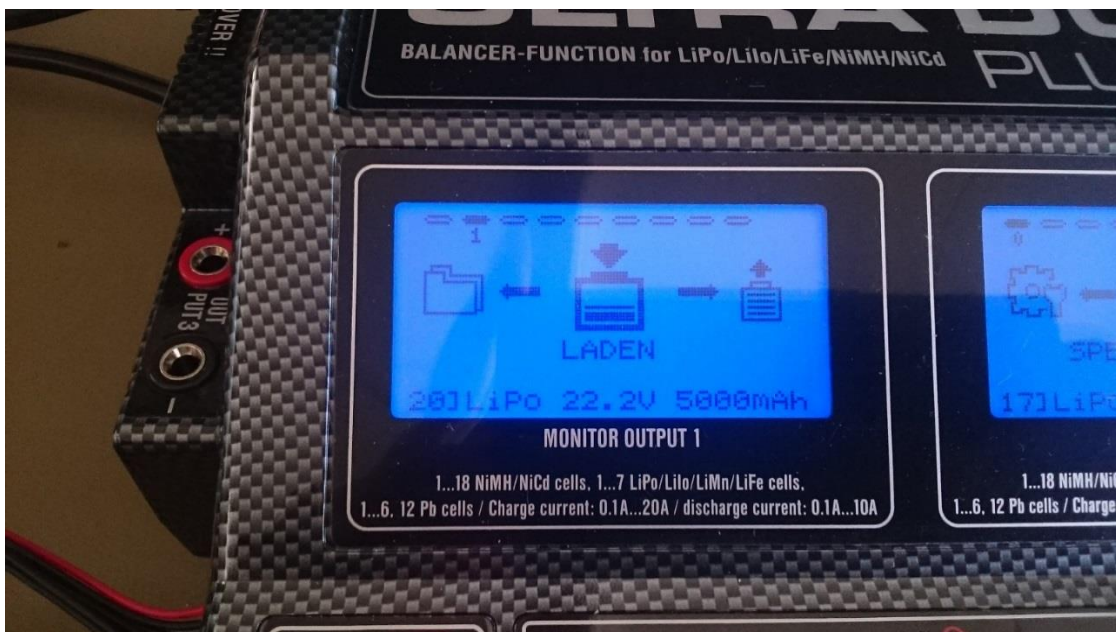


Figure 36: Battery charging status icon

Use the CC/CV mode in the next screen (CC/CV is the common charging mode)

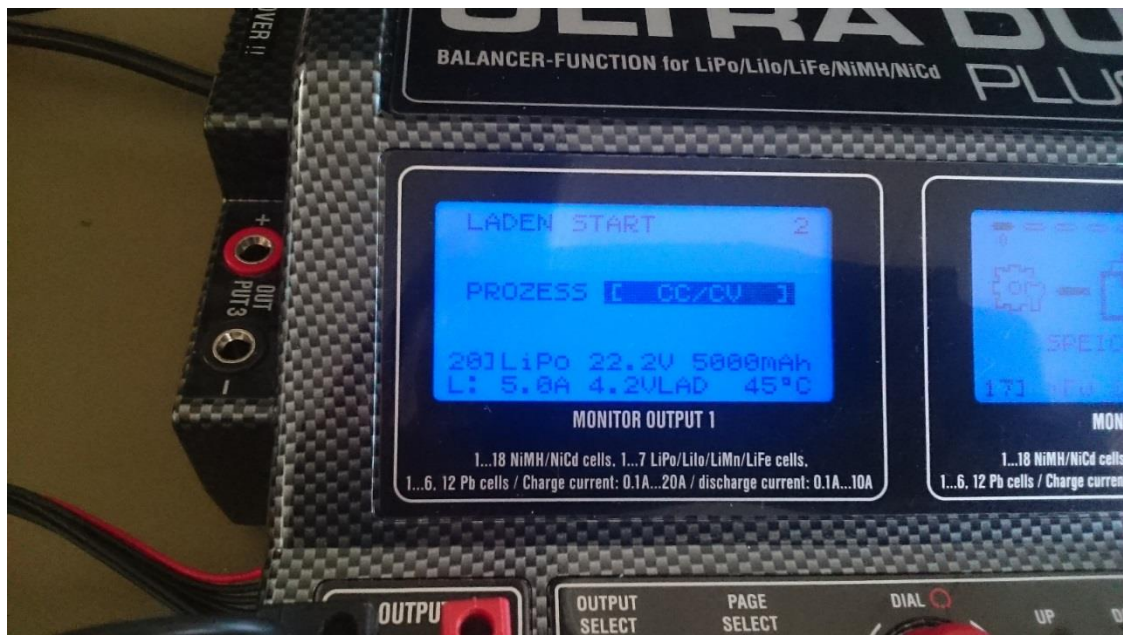


Figure 37 charge mode

Set delay time to (VERZ. ZEIT) "AUS" (off)

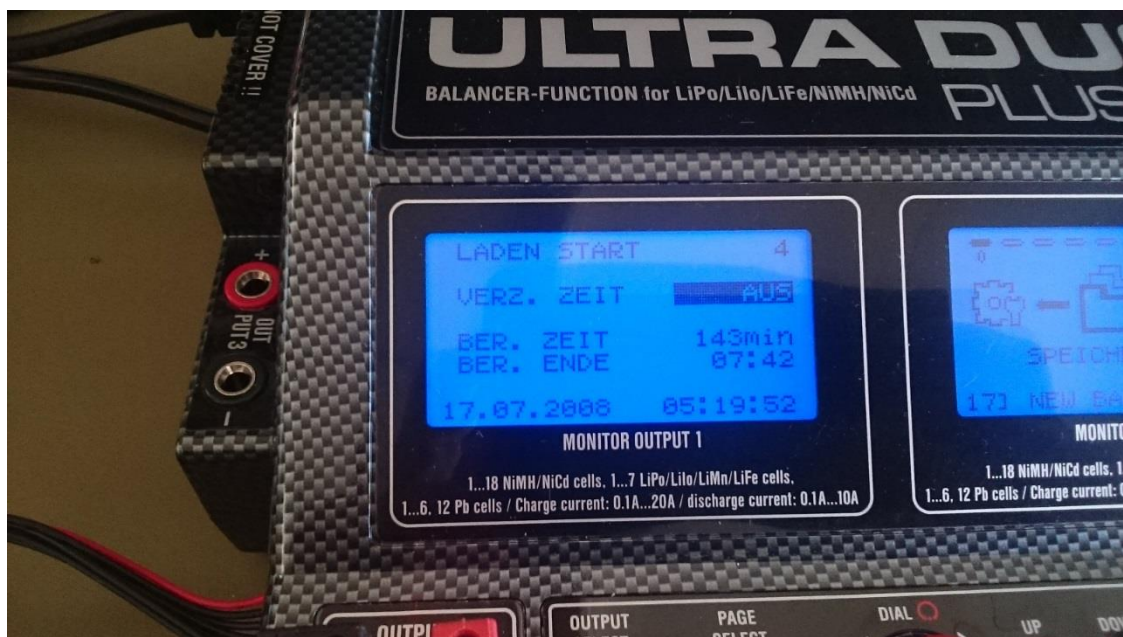


Figure 38 charge start screen

Wait until connection is checked



Figure 39 connection check

If Balance Board is connected: check cell count and confirm

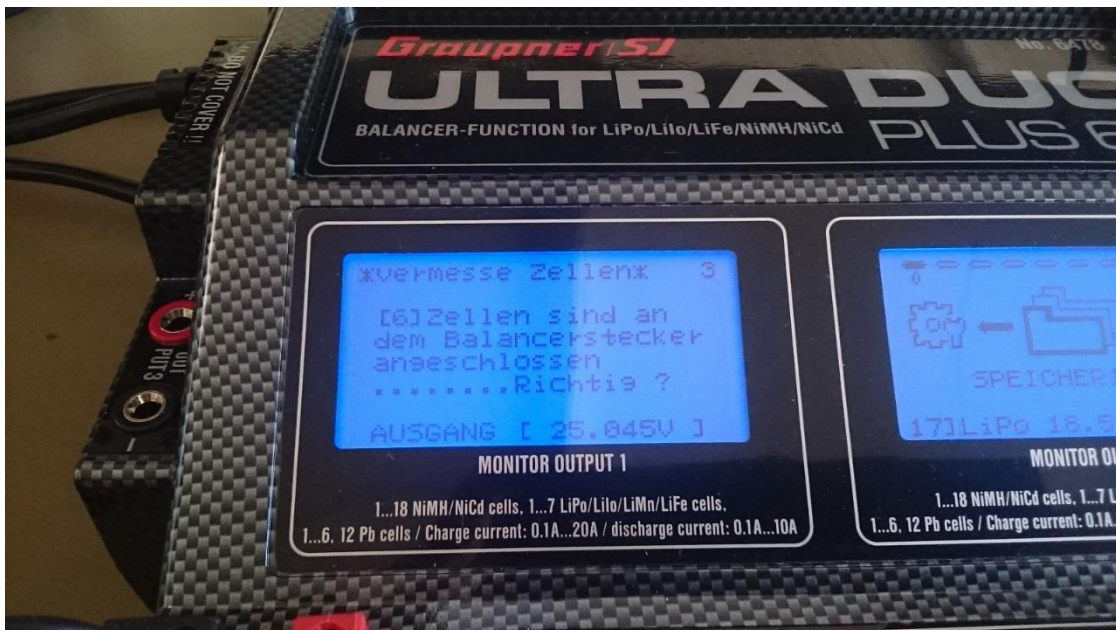


Figure 40 cell check screen

The power station starts charging and shows you current values



Figure 41 charging screen

Turn the RCW to see more information

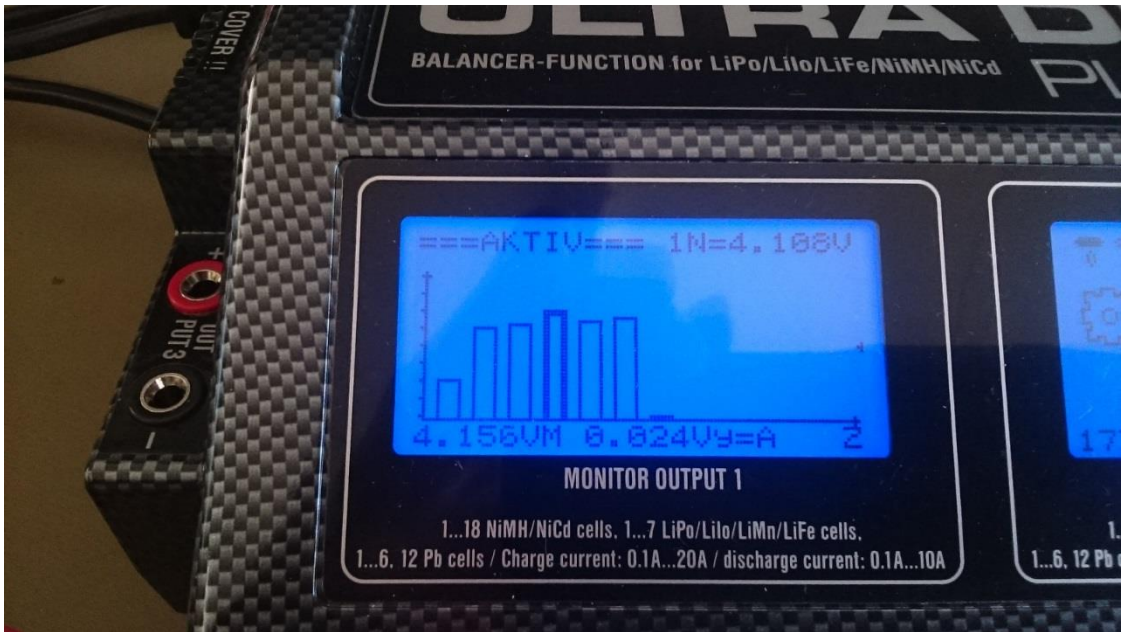


Figure 42 cell load

To configure the charge options press and hold RCW on “LADEN”-item until you enter der “LADE KONFIG.” settings



Figure 43 charge config

BALANCE BOARD: plug the balance board into the connector on left for channel 1 or right side for channel 2



Figure 44 balancer plug

Plug the cell cable from battery to the 7S adapter – use the plug with the right count of cells

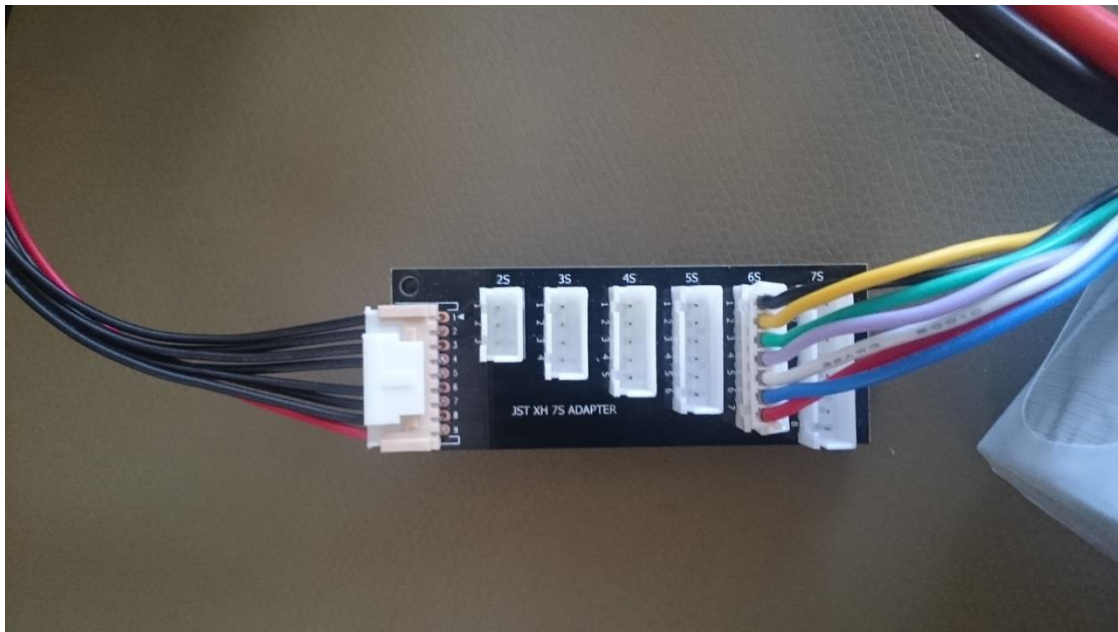


Figure 45 balancer adapter

16 BIBLIOGRAPHY

- [1] C. D. H. S. Y. C. Löser, "Project BumbleBee," \\documentations\Bumblebee_Documentation\Final_Report.pdf, 2015.
- [2] Wikipedia, "Wikipedia: Scrum," [Online]. Available: [https://en.wikipedia.org/wiki/Scrum_\(software_development\)](https://en.wikipedia.org/wiki/Scrum_(software_development)). [Accessed Januar 2016].
- [3] live-hobby.de, "www.live-hobby.de/," [Online]. Available: <http://www.live-hobby.de/out/pictures/master/product/4/33116.jpg>. [Accessed 19 1 2016].
- [4] Graupner GmbH & Co KG, "www.graupner.de," [Online]. Available: http://www.graupner.de/fileadmin/downloadcenter/anleitungen/20070222092922_4701_GB.pdf. [Accessed 11 1 2016].
- [5] Microsoft, "Microsoft Developer Network," [Online]. Available: msdn.microsoft.com. [Accessed 11 1 2016].
- [6] F. Seifert, "Konzeption und Realisierung einer Steuerrechner-Plattform für ein Quadrocopter Flugmodell," Ulm, 2013.
- [7] uClib.org, "Buildroot," [Online]. Available: <https://buildroot.uclibc.org/>. [Accessed 7 1 2016].
- [8] M. H. Andreas Klinger, "Embedded-Linux-Systeme mit Buildroot erstellen," [Online]. Available: <http://www.elektronikpraxis.vogel.de/themen/embeddedsoftwareengineering/implementierung/articles/173855/>. [Accessed 7 1 2016].
- [9] qgroundcontrol.org, "qgroundcontrol.org," [Online]. Available: http://qgroundcontrol.org/dev/mavlink_linux_integration_tutorial. [Accessed 10 1 2016].
- [10] QGroundControl, "qgroundcontrol.org," [Online]. Available: http://qgroundcontrol.org/_detail/screenshots/fly.png?id=start. [Accessed 19 1 2016].
- [11] prabhudesai, "https://xplorenlearn.files.wordpress.com," [Online]. Available: https://xplorenlearn.files.wordpress.com/2014/11/yaw_pitch_explain.png. [Accessed 19 1 2016].
- [12] D. Brunner. [Online]. Available: http://www.technik-consulting.eu/Optimierung/Quadrocopter_PID-Regelung.html. [Accessed 11 1 2016].
- [13] Analog Devices , "www.sparkfun.com," [Online]. Available: <https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf>. [Accessed 7 1 2016].
- [14] Honeywell, "www.sparkfun.com," [Online]. Available: <http://cdn.sparkfun.com/datasheets/Sensors/Magneto/HMC5883L-FDS.pdf>. [Accessed 7 1 2016].
- [15] InvenSense, "www.sparkfun.com," [Online]. Available: <https://www.sparkfun.com/datasheets/Sensors/Gyro/PS-ITG-3200-00-01.4.pdf>. [Accessed 7 1 2016].
- [16] Buildroot developers, "The Buildroot user manual," Buildroot, 10 01 2015. [Online]. Available: <http://buildroot.uclibc.org/downloads/manual/manual.html>. [Accessed 19 01 2016].
- [17] Graupner GmbH & Co KG, "graupner.de," [Online]. Available: http://www.graupner.de/mediaroot/files/6478_ULTRA%20DUO%20PLUS%2060_en.pdf. [Accessed 5 1 2016].
- [18] DJI Innovations, [Online]. Available: http://download.dji-innovations.com/downloads/nazam-v2/en/NAZA-M-V2_Quick_Start_Guide_en.pdf. [Accessed 19 1 2016].
- [19] cypress.com, [Online]. Available: [http://www.cypress.com/fckImages/myresources/USBControllers_Overviewimg\(1\).jpg](http://www.cypress.com/fckImages/myresources/USBControllers_Overviewimg(1).jpg). [Accessed 19 1 2016].

