LukOef

Hochschule Ulm

FINAL REPORT

X-Copter

Table of contents

[2 Project Description 4](#_Toc440284531)

[3 Project members 5](#_Toc440284532)

[4 Analysis of customer needs 6](#_Toc440284533)

[4.1 Old Definition of Requirements: 6](#_Toc440284534)

[5 Customer Needs 7](#_Toc440284535)

[6 Project Management 8](#_Toc440284536)

[6.1 Scrum 8](#_Toc440284537)

[6.2 Roles in Scrum 8](#_Toc440284538)

[6.2.1 Product Owner 8](#_Toc440284539)

[6.2.2 Development Team 8](#_Toc440284540)

[6.2.3 Scrum Master 8](#_Toc440284541)

[6.3 Sprint 1 8](#_Toc440284542)

[6.4 Sprint 2 //TODO after sprint 2!? 10](#_Toc440284543)

[6.5 Milestone 3 //TODO 10](#_Toc440284544)

[7 Interim Evaluation by the team 11](#_Toc440284545)

[8 Project Documentation 12](#_Toc440284546)

[8.1 System Architecture //TODO 12](#_Toc440284547)

[8.2 First Test flight // TODO zu viele kleine abschnitte…viel mehr beschreiben 13](#_Toc440284548)

[8.2.1 Attempt 1 13](#_Toc440284549)

[8.2.2 Attempt 2 13](#_Toc440284550)

[8.2.3 Attempt 3 13](#_Toc440284551)

[8.2.4 Attempt 4 13](#_Toc440284552)

[8.2.5 Conclusion 13](#_Toc440284553)

[8.3 Second Flight Test // TODO noch schlimmer 14](#_Toc440284554)

[8.3.1 Organisation 14](#_Toc440284555)

[8.3.2 Changes at the XCopter since the last Flight test 14](#_Toc440284556)

[8.3.3 Attempt 1 14](#_Toc440284557)

[8.3.4 Conclusion 14](#_Toc440284558)

[9 Hardware 15](#_Toc440284559)

[9.1 Power supply 15](#_Toc440284560)

[9.1.1 Final Solution 15](#_Toc440284561)

[9.2 Remote Control 16](#_Toc440284562)

[9.2.1 Choice of the RC – Controller and the receiver 16](#_Toc440284563)

[9.2.2 RC Transmitter Settings 16](#_Toc440284564)

[9.3 Commercial Flight control 17](#_Toc440284565)

[9.4 Kinect and USB Host Controller Documentation 18](#_Toc440284566)

[9.4.1 Initial Situation 18](#_Toc440284567)

[9.4.2 Requirements for the USB-Controller 18](#_Toc440284568)

[9.4.3 Common USB-Controller Packages 18](#_Toc440284569)

[9.4.4 Controllers that come into question 18](#_Toc440284570)

[9.4.5 Conclusion 20](#_Toc440284571)

[10 Linux System 21](#_Toc440284572)

[10.1 Reasons for using a custom embedded Linux 21](#_Toc440284573)

[10.2 Build Root 22](#_Toc440284574)

[10.3 Wi-Fi Connectivity 22](#_Toc440284575)

[10.3.1 Implementing the Driver 22](#_Toc440284576)

[10.3.2 Configure the Wi-Fi connection 23](#_Toc440284577)

[10.3.3 Result 24](#_Toc440284578)

[10.4 MCAPI 24](#_Toc440284579)

[10.4.1 MCAPI Entities 25](#_Toc440284580)

[10.4.2 Packet Channel 25](#_Toc440284581)

[10.4.3 MCAPI example program 25](#_Toc440284582)

[10.5 MAVLink Communication Protocol 25](#_Toc440284583)

[10.5.1 MAVLink on Linux 26](#_Toc440284584)

[10.6 QGroundControl 26](#_Toc440284585)

[10.6.1 Monitoring 26](#_Toc440284586)

[11 Flight Controller 27](#_Toc440284587)

[11.1 Flight Controller basic concept and structure 27](#_Toc440284588)

[11.1.1 What is a Flight Controller? 27](#_Toc440284589)

[11.1.2 Basic concept of a flight controller 27](#_Toc440284590)

[11.1.3 Structure of the XCopter flight controller 28](#_Toc440284591)

[11.1.4 Dataflow through the flight controller 29](#_Toc440284592)

[11.1.5 Programm flow of the flight controller 30](#_Toc440284593)

[11.1.6 Tasks and Timing 33](#_Toc440284594)

[11.2 Motor PWM Signal 33](#_Toc440284595)

[11.3 Interfaces between components 33](#_Toc440284596)

[11.4 RC-Receiver 34](#_Toc440284597)

[11.4.1 Graupner HoTT-SUMD-Signal Definition 34](#_Toc440284598)

[11.4.2 Structure of a HoTT- SUMD frame 34](#_Toc440284599)

[11.4.3 SUMD\_Header section description 34](#_Toc440284600)

[11.4.4 SUMD Data section description 35](#_Toc440284601)

[11.4.5 SUMD\_CRC section description 35](#_Toc440284602)

[11.4.6 Channel data interpretation 35](#_Toc440284603)

[11.4.7 Implementation of the SUMD Parsing 35](#_Toc440284604)

[11.4.8 Saving raw SUMD-Frame Bytes from the UART 35](#_Toc440284605)

[11.4.9 Interpreting the received SUMD-Frame 35](#_Toc440284606)

[11.5 UART Driver 35](#_Toc440284607)

[11.5.1 SUMD-Frame-high 36](#_Toc440284608)

[11.5.2 SUMD-Frame-low 36](#_Toc440284609)

[11.6 Sensor Data Manager (SDM) 36](#_Toc440284610)

[11.7 Sensor Data Filter 36](#_Toc440284611)

[11.8 PID Regulators 37](#_Toc440284612)

[11.8.1 General 37](#_Toc440284613)

[11.8.2 PID Regulators in the simple flight controller 38](#_Toc440284614)

[11.9 PID to motor mapper 39](#_Toc440284615)

[11.9.1 The idea of the PID to motor mapper module 39](#_Toc440284616)

[11.10 SparkFun Sensor Board 40](#_Toc440284617)

[11.10.1 Calibration of the accelerometer 40](#_Toc440284618)

[11.10.2 Calibration of Gyro meter 41](#_Toc440284619)

[11.11 Logging 41](#_Toc440284620)

[12 Lesson Learned //TODO 100 mal zu wenig 42](#_Toc440284621)

[12.1 General problems with the Flight Controller 42](#_Toc440284622)

[12.2 Scrum 43](#_Toc440284623)

[12.3 Long term calendar management 43](#_Toc440284624)

[12.4 Small tasks take more time than thought 43](#_Toc440284625)

[13 Howto Section 44](#_Toc440284626)

[13.1 ARM / Linux: VM, Cross compile, execute code 44](#_Toc440284627)

[13.1.1 Virtual Machine usage and folder structure 44](#_Toc440284628)

[13.1.2 Cross compiling code on Ubuntu for ARM 45](#_Toc440284629)

[13.2 Calibration of the accelerometer 46](#_Toc440284630)

[13.3 Charging Batteries for the XCopter 47](#_Toc440284631)

[13.3.1 Charging Batteries: 47](#_Toc440284632)

[13.3.2 Step-by-Step-Solution 48](#_Toc440284633)

[14 Literaturverzeichnis 56](#_Toc440284634)

# 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 on the ground.

# Project members

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

# Analysis of customer needs

## Old Definition of Requirements:

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

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

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

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

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

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

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

# Customer Needs

Since this is an ongoing project, the basic hardware functionality and the interprocessor communication was implemented by the previous Team (Team Bumblebee). The customer needs of Team Bumblebee are documented in their Final Report ( [1] Seite 5, “Analysis of Customers 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
* Visualisation 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".
* µCOSII as real time operating system for NIOSII-CPUs (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)

# Project Management

## 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 they 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 to deliver 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 having regard to the dependencies of the items and the priority when they are needed. After the sprint planning meeting was 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” every day. 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. At this moment it is necessary that every team member also talk honestly about issues they 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 note which planned tasks are completed and which ones not. At the end of the meeting there is a short demonstration of the completed tasks for the stakeholders. This is also the point 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 that is without the stakeholders. In this meeting the scrum team talks about the previous sprint and the problems they occurred during this sprint. Here the team can also talk about personal problems like worse communication between different team members or missing resources. The goal of this meeting is to improve things for the following sprint. After the sprint review the next iteration starts again with the sprint planning meeting. Because one of the key principle of scrum is its agile switch of customer changes during a project and because scrum is a step by step developing methodology the decision was made to use scrum. [2]

## Roles in Scrum

### Product Owner

The product owner is close 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]

### Development Team

The development team is self-organizing in Scrum. A team is made up of 3-9 members. The team is responsible for the progress of the project. Each team had their own tasks.   
In each task the actual work is described by the product owner and scrum master. If the development team finishes some tasks they continuing with the next open task. [2]

### Scrum Master

The scrum master is coaching the team with the scrum principles. He is responsible to remove impediments of the development team. The scrum master facilitation team events like the daily scrum or other meetings. He acts as a buffer between the team and the customer. [2]

## Sprint 1

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

1. Select the remote:

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

1. Stable battery

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

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

* programming the charger
* extern power supply

1. USB-Controller

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

1. Commercial flight controller

* get the Software and install it
* feature list
* configure the cruise [control](http://www.dict.cc/englisch-deutsch/control.html)
* connect the cruise controll to the rotors
* connect the cruise controll to the flight controller
* connect the flight controller to the XCopter
* get a connection to the remote
* configurate the flight controller
* configurate 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 were the team organized all Scrum activities. The X-axis descripes 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 statrs 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 handle most of the task in one month. In the middle of may there was a task that needs more time then expected. It was the USB-Controller which we don’t completed in the first Sprint. After all the first assessment was very good and most of the task were finished at the scheduled time

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

1. Commercial flight controller

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

1. Simple flight controller

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

1. USB-Controller

* Search for a new USB-Controller

1. Day of informatics

* Make a presentation about the XCopter

Make a poster for the day of informatics

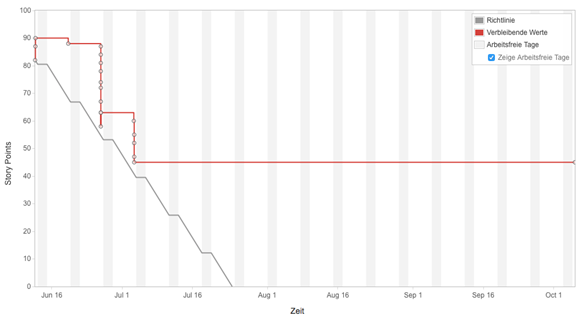


Figure 2: Burndown-Chart Sprint2

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

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

For safety reasons it was not possible to test the XCopter in the laboratory or at the campus. So for testing it was neccessary to go to a Model flying site. In addition the person that controlled had to had a Model Flying insurance. Jochen Höft already had an 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, a basic equipment of tools and spare parts was taken to the Model flying site.

### Attempt 1

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

### Attempt 2

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

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

### Attempt 3

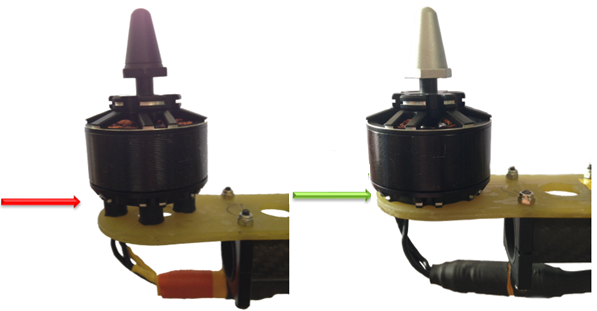
This attempt started well but the XCopter lost upward boost at one side and crashed. Error Analysis: It turned out that one rotor loosened. All rotor screws were tightened.

### Attempt 4

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

### Conclusion

The XCopter vehicle should be able to fly. The rubber vibration damper is not recommended, because the motors apply to much strength on them and finally they get broken (Ref image unten). Another Test is necessary. Before a flight it’s neccessary 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.



## Second Flight Test //TODO EINPFLEGEN INS HAUPT DOC

### Organisation

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

### 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 flight. In the future it could be possible that the cameras (which should be mounted to map the surroundings) has to be fixed with dumpers in case of the vibration. In other self-made quadrocopter 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 the further procedure of the project. In the further steps the team can concentrate on the main aspects of the project.

# Interim Evaluation by the team

After a half of the project some important milestones like the first successful flight or the successful transmission from the RC transmitter to the NiosII-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. So 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 member 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 task 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 the section (ref Stephan scrum problems).

## 

# 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 is 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 will be replaced by two ASUS cameras that will be connected to the USB controller on the DE1-SoC board.   
It was also decided to connect the XCopter to the ground station, which is running on a PC or Laptop, via WLAN. Therefore a USB WLAN module was used and connected to the DE1-SoC (see chapter 10.3).  
The other architecture specifications didn’t really change. The DE1-SoC board contains three CPUS, an ARM9 Dual Core that handles the coordination, and two NIOS II CPUs. One is responsible for collision detection and the other CPU for the flight controller. Each of those CPUs communicates with the other processors via the MCAPI (see chapter 10.4). An accumulator pack powers the motors and the DE1-SoC. 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 (see chapter 11).   
More details on the specific components can be found through this document at the respective chapters.

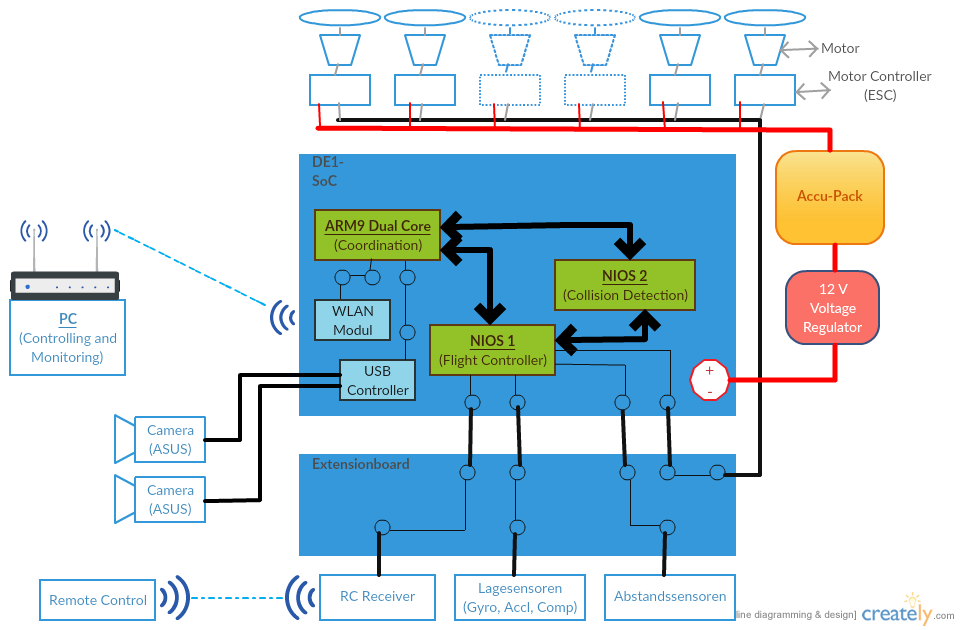


Figure 3: System Architecture of the XCopter

# Hardware

## Power supply

Because the power supply that was available from the older BumbleBee-Project, isn’t working, the need for a new, functioning power supply came up.  
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 on the XCopter and the SoC-Board restrictions, which are 12V input voltage and 3.5A output current. For provision 1.5A was added to the I\_out parameter.

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

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

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

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

### Final Solution

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

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



## Remote Control

### 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)
* Provide 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 standart
* Graupner provides lots of datasheets for each product
* Graupner has a big RC- community

#### Major properties of the RC- Controller:

* 8 channel
* 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

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

### 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 behaviour [Motor an K1] | "kein" (default) |
| changing Flight Phase with Channel8 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 [Grundeinst]

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

Table 2:RC Receiver Settings [Telemetrie]

More information is available in the original Graupner MX-16 user manual ( [2])  
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

## 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 control. In this case the DJI NAZA V2 was used.

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

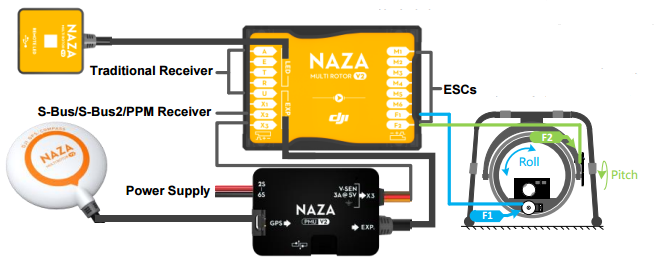


Figure 4 NAZ V2

Overview of DJI NAZA V2

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

## Kinect and USB Host Controller Documentation

### Initial Situation

The customer wants two Kinects to be put on the 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 colour camera. For proper 3D-Image data colour- and depth camera have to work at the same time and cannot be separated, which strikes out the option to save bandwidth with using only one camera at the time.

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

### Requirements for the USB-Controller

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

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

### Common USB-Controller Packages

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

### Controllers that come into question

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

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

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

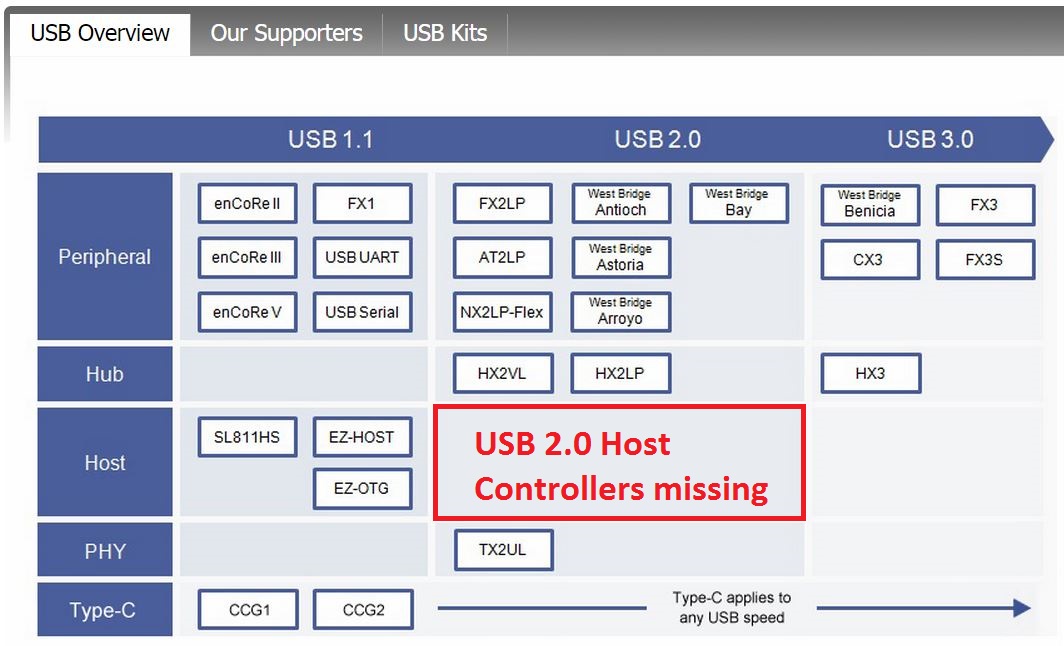


Illustration 1: Cypress USB offerings (http://www.cypress.com/fckImages/myresources/USBControllers\_Overviewimg(1).jpg)

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

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

USB Controller Comparison

### Conclusion

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

# Linux System

## 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 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 weightiness 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 drones. With a stripped down OS there are less attack vectors for hackers 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.

## Build Root

Buildroot is an open source project that makes it possible to create an individual Linux system. It’s divided in different parts. Buildroot can automatically build the required cross-compilation toolchain, create a root file system, compile a Linux kernel image and generate a boot loader.  
Busybox and uClibc are the main parts of Buildroot. uClibc is a standard C-Library for embedded Linux systems. Busybox is a program which includes all necessary Unix services in one compressed package. It’s perfect to use for embedded systems with only limited resources. The easiest way to implement new drivers into the ARM Linux system is to do it with buildroot. Cross-compiling drivers manually is very difficult and leads to a lot of errors. Even if there is a driver available for an ARM Cortex A9 processor 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-loader. Its task is to boot up the Linux system. Usually these boot loaders has to be configured once and not at any time when something has been changed in the Linux image.   
The main parts of the Linux system was built by Mr. Strahnen. The system runs on kernel 3.10.37, because of the long term support. It’s not possible do compile code on the ARM CPU because of the missing toolchain. Therefore it’s necessary to cross-compile the code on the virtual machine. (14.1) Components like the WIFI driver were added via Buildroot. To increase the performance of the system it’s possible to deactivate the unused packages.  
[4] [5]

## Wi-Fi Connectivity

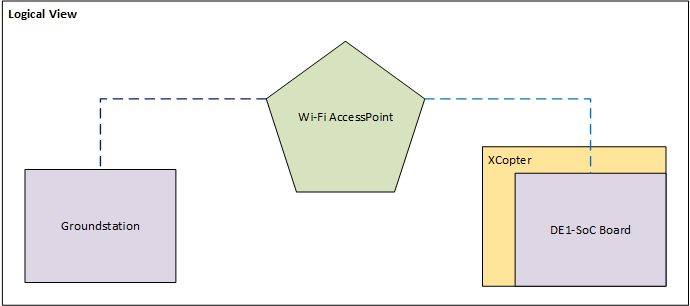


Figure 5: The logical view of the Wi-Fi components

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 RTl8188 chip) plugged in the XCopter DE1-SoC board (DE1). Both are connected with an access point (AP) (Figure 4: The logical view of the Wi-Fi components).

### Implementing the Driver

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

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

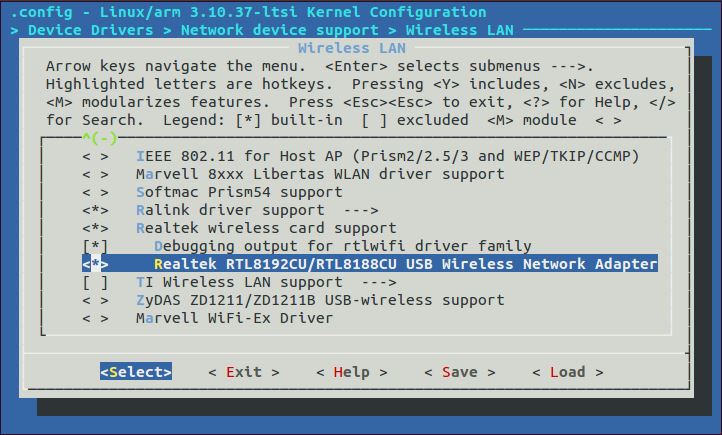


Figure 6: Buildroot driver selection

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 5: Buildroot driver selection). Now the driver file can be loaded successful but there was an error with a missing firmware file. Adding the right firmware in Buildroot solved this issue. Now the basic settings are set and the DE1 can establish a wireless connection to the AP.

### 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 has to be set up following changes (Figure 6: Interface configuration file):

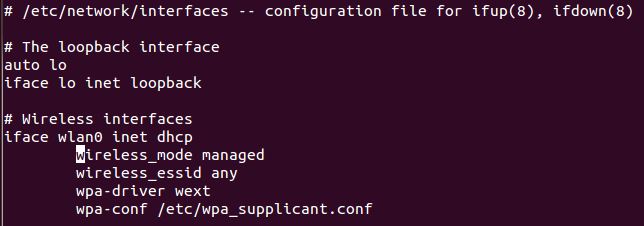


Figure 7: Interface configuration file

The wpa\_supplicant (Figure 7: wpa\_supplicant file) file needs the next showing entries to work.

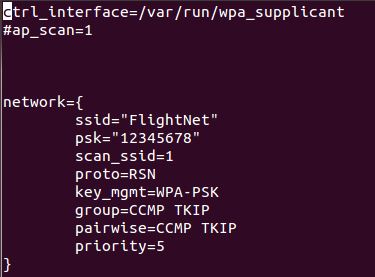


Figure 8: wpa\_supplicant file

To get an IP lease from AP 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 do at the start up. Making a new file in the */etc/init.d/* folder with the file name *S50start\_wlan* and insert the content of Figure 8: Content of the start-up script.

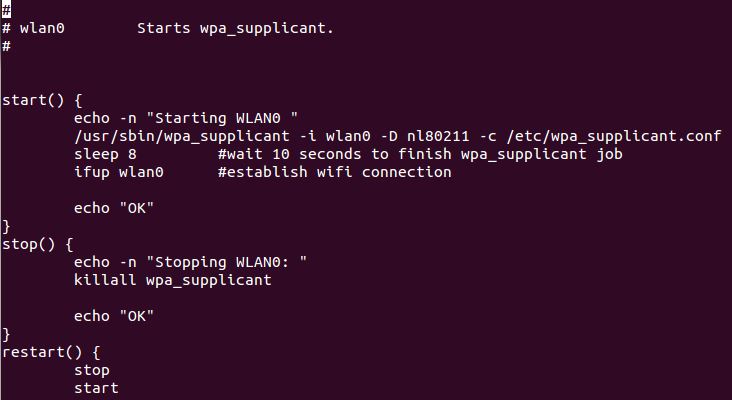


Figure 9: Content of the start-up script

### 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-authorization access. The Wi-Fi connection was successful 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.

## 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 NIOS1. (For a more detailed look of the system architecture see chapter 8.1) The NIOS1 CPU, which holds the flight controller, sends the logged telemetry data to the ARM CPU. MCAPI is necessary for this communication between the CPUs.

In the following, the MCAPI basics are explained.

### MCAPI Entities

For the MCAPI there are three basic entities: domains, nodes, and endpoints. Next a short description of the tree entities.

#### Domain

A MCAPI domain gathers several MCAPI nodes. One domain could have more than one domain.

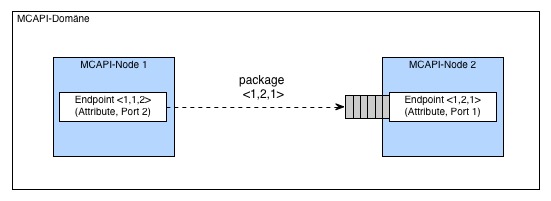
#### Node

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

#### Endpoint

A MCAPI endpoint is a socket like communication. 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.

### 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. Packet channel data transfer

### MCAPI example program

In the VM is a folder l\_MCAPI. There are two c files, one for the NIOS and 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 to display log data on the ground control.

## MAVLink Communication Protocol

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

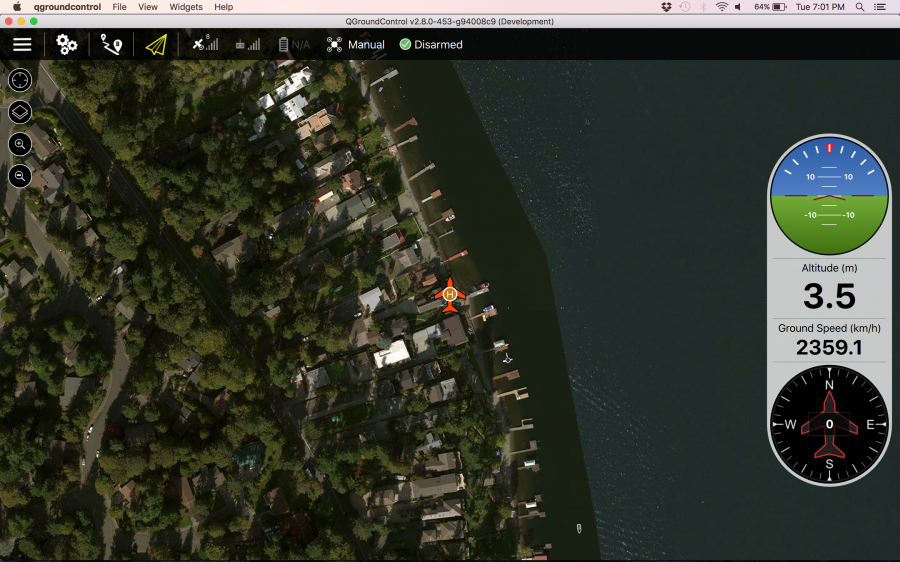
### MAVLink on Linux

To send data from the ARM processor to the ground station, the example program for Linux integration on the MAVLink website [6] was used. This program tests the UDP connection to QGroundControl. The program sends the necessary MAVLink packets to the QGroundControl which repeat with a heartbeat. The program can be compiled on Linux without changes. We only have to change the IP address from localhost to the IP of the target PC where QGroundControl is running.

## QGroundControl

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

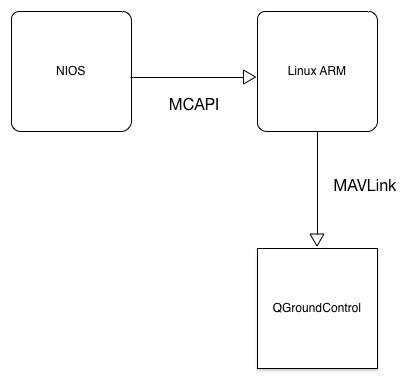
* Open-source
* Widows / Linux / MacOS support.
* 2/3D aerial maps
* In-flight manipulation of waypoints
* Real-time plotting
* Loggin of sensor logs
* Supports multiple Autopilots
* MAVLink protocol support



QGroundControl http://qgroundcontrol.org/\_detail/screenshots/fly.png?id=start

### Monitoring

Combining all the communication protocols there is a chain that the XCopter ist used for monitoring. Its starts by logging data on the NIOS side like values of the sensors. To visualize this data the NIOS CPU sends his data to the Linux via MCAPI. The Linux pack this data in da MAVLink struct and sends his packet over a socket to the QGroundControl. QGroundControl visualize the data and monitor them on any PC were QGroundControl runs. The following charts shows the communication chain for a better comprehension.



QGroundControl communication chain

# Flight Controller

## Flight Controller basic concept and structure

### 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 9.4. 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.

### 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 the UAV is in the air: With the data of the sensors, the pitch, yaw and gear orientation (see REF Figure 9) of the UAV is calculated.

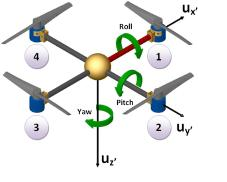


Figure 10: Roll, Pitch and Yaw orientation [https://xplorenlearn.files.wordpress.com/2014/11/yaw\_pitch\_explain.png]

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 11.10. Depending of the output of these PID’s the individual motors have to run at different speeds to stabilize the UAV in the air.

### Structure of the XCopter flight controller

In the XCopter project, the flight controller is separated in 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 in a later section 11.1.6. You can see the program flow in more detail in the chapter 11.1.5

In REF Figure 10 a component diagram, that shows the structure of the flight controller, can be seen. Please note 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 11.8 The ‘RC Receiver’ is colored in green and is elucidated in section **“RC Receiver”.** The main controlling part contains multiple components, is colored in yellow. Its particular components are described separately in later sections, whereas the functionality will be pointed out in the following section.

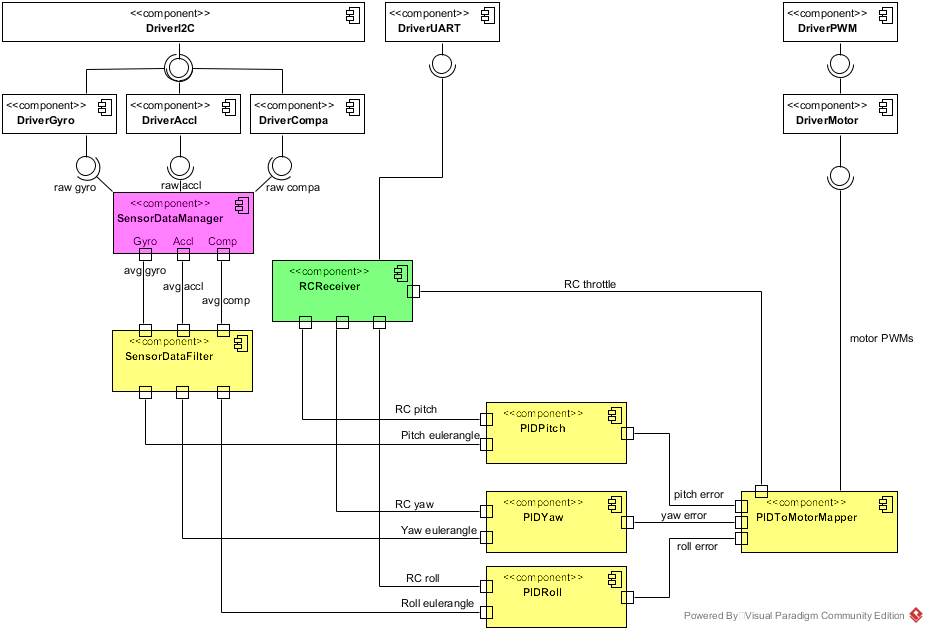


Figure 11: Flight Controller structure

### Dataflow through the flight controller

Basically, what a flight controller does, is getting data from the sensors and the remote control and translate this into motor revs. Or the other way round: The motors are running at different velocities depending on the data it gets from the user (remote control) and the sensors. Therefore, understanding the dataflow is very important because it is also 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, please see section 11.5. More details about the different components can be found in their respective chapters.

In Figure 11 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.

Sensors

Remote Control

Sensor Data Manager

Sensor Data Filter

PIDs

PID To Motor Mapper

Motors

averaged sensor data

raw sensor data

euler angles

PID error values

motor PWM signals

euler angles

throttle

Figure 12: Dataflow through the flight controller

### Programm flow of the flight controller

An overview of the program flow and algorithm of the flight controller can be seen in REF Figure 12 and is described in the following. More details on the implementation can be found in the REF **“Implementation of the Flight Controller”.**

When the XCopter is turned on, the program starts with an initialization progress. At first the, hardware device drivers, such as I²C and UART drivers 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 SensorDataManager and RCReceiver 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 SensorDataManager task, which is colored in pink again, 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. Then, 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 RCReceiver task. The main loop of this task reads the incoming SUMD frames that the remote control transmits and CRC-checks them for validation. If the frame is valid, the RC values are scaled and converted into euler angles. When this is done, the RCReceiver task notifies the main task in the same way the SensorDataManager 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 and roll euler angles of the XCopter. After storing the filtered data, it is checked if there are new remote control values available from the RCReceiver 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.

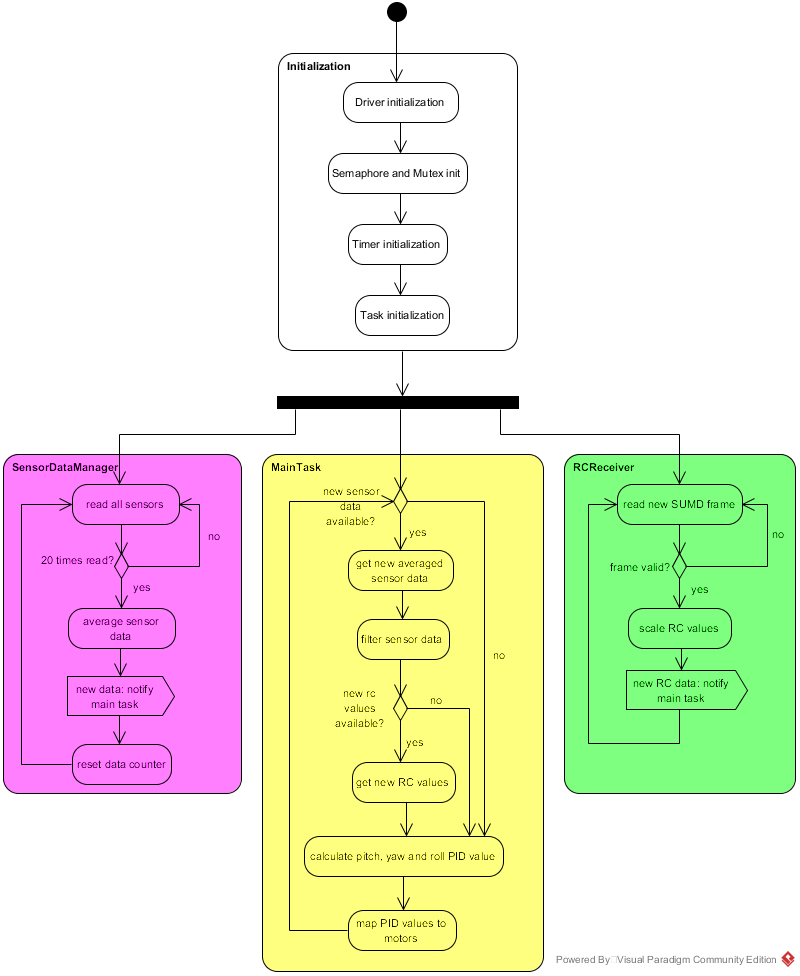


Figure 13: Program flow of the flight controller

### Tasks and Timing

All tasks of the XCopter flight controller have to run in a specific hierarchy.  
The used version of the operating system *uC/OS II* has no built-in function, which is precise enough  
to make all tasks periodic in an appropriate way.  
However the A*ltera 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 has a resolution of 1µs.  
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 in below (Figure 1).

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

**}**

## 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 50 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 and 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. But it is possible to calibrate the ESC’s so we made them working in a range from 8 to 218. Now the resolutions is fine enough and the ESC’s are working well.

## Interfaces between components

In the table below (Ref Table In- and output ranges of Components) shows the interfaces between the components.  
Notes:

\* These values are not determined jet, however they 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. For more information see (12.2).  
\*\* These values depend on the P, I and D values in the PID’s. Since these weren’t finally set jet, the value ranges aren’t defined. For more information see (Ref zur Problemsection über das PID)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 3: Input and output ranges of the components

## 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.

### 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 50µs between transmitted data bytes. The serial connection has to be setup whit the following:

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

### Structure of a HoTT- SUMD frame

A single SUMD data frame 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.

### SUMD\_Header section description

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

### SUMD Data section description

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

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

### SUMD\_CRC section description

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

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

### Channel data interpretation

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

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

### Implementation of the SUMD Parsing

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

- 115200 Baud

- No Parity

- 1 Stop Bit

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

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

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

### Interpreting the received SUMD-Frame

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

## UART Driver

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

### SUMD-Frame-high

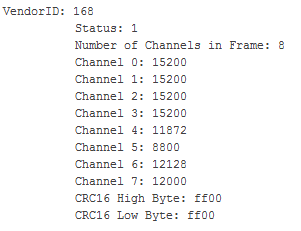


Figure 14 SUMD-Frame-high

### SUMD-Frame-low

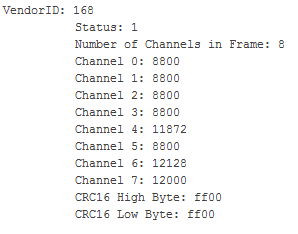


Figure 15 SUMD-Frame-low

## Sensor Data Manager (SDM)

The Sensor Data Manager 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 is 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) were 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 value. Each of the 3 sensors provides 3 values, one for each axis(x,y,z). So the array providing space for 20 (=VALUE\_NUM) samples.   
Additionally there is and 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 interface between the Sensor Data Manager task and the Main Controlling task.   
The gyro meter provides a temperature value which could in theory used to compensate the changing value offset while warmup but this is not implemented jet.

The way to go to access the sensor data is check the SDM\_NEW\_DATA\_AVAILABLE-flag and if it’s set call the getSensorData(.. , ..). This functions has 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 int for the deltaT.

## Sensor Data Filter (SDF)

This component 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 (REF [1], [2]). It was configured 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 REF **“Calibration of accelerometer”.**

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

* A pointer ‘int16\_t\* avgSensorData’ on the array of the averaged sensor data from the sensor data manager. This is considered as an input parameter.
* A pointer ‘float\* filteredSensorData’ on the array where the result (filtered data) is saved to. This is considered as an output parameter.
* A number ‘uint32\_t averagedDataDelatT’, 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. For more details of the program flow please see chapter REF **“Program flow of the flight controller”.**

[1] \ XCopter\code\SimpleFlightController\filterungHerrSteiper\ Duocopter\_PAP\_Report.pdf

[2] \XCopter\code\SimpleFlightController\filterungHerrSteiper\FreeIMU\_raw\_extendef\_6\_ethernet\_udp\_3\_test\ FreeIMU\_raw\_extendef\_6\_ethernet\_udp\_3\_test.ino

## PID Regulators

### General

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

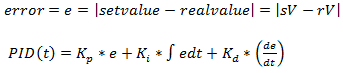


Figure 16 PID-Formula

#### The Proportional Part

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

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

and it will return to the motor.

#### The Integral Part

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

#### The Differential Part

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

### 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 11.10every 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 RCReceiver module are providing the set points, and the returned values of the SensorDataFilter are taken for the real values of the PIDs (11.9.1).



Table 4: 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 11.10.

The constants Kp, Ki, Kd (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. This 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 ( [7]). 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.

## PID to motor mapper

### The idea of 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 this three output values as well as the throttle value of the RCReceiver module for calculating the PWM signal for each motor. This is the task of the PIDToMotorMapper 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 get 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 that the boost which is needed to balance out the Xcopter (the boost that is generated by the PID values) 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” didn´t get over 100%.

#### Explanation of the mapping table

The following extract shows the mapping table of the PIDToMotorMapper module. The function PIDMIX gets a throttle value from the RCReceiver 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.



#### Explanation how to ensure the correct boost of the PIDs

The second problem which is described in section 12.1 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 which 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 couldn´t go up anymore. The following formula is used in a while loop in the motor mapper module to decrease the value of and to adjust the value of .

and are the constants which were used to adjust the and the . The two constants are both initialized with one and in case that the value of is greater than the value of , will be decremented with 0.1 and will be adjusted with the formula:

The following code snippet computes the values for the (throttleMix) and (pidMix), checks if “pidMix” is bigger than the throttle value “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.



The following function ‘*computeCThrottle(…)’* computes the value for the constant.



After calculating the PWM signals for the motors the function “writeToMotors()” is called which sets the motor speed for each one. The motor speed is indicated in percent. The mapper module is not validated at the moment. Because the constants for the PIDs not available yet. More information about can be read at the section ( REF problems / future work motor mapper)

## 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 [2] for exact information. Each sensor returns a value for the x, y, and z axis. Because each axis of the accelerometer and the gyroscope have a different range of values it is necessary to convert the output range into a uniform range. Only then it is possible to use the sensor values in the following modules of the flight controller. So the accelerometer and the gyroscope must be calibrated.  
[8] [9] [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. This values are needed to convert the output range of the sensor into a uniform range.

To get the offset value and the scale value for each axis the maximum, minimum and the zero point values are needed. The How-to “Calibration of the accelerometer” 13.2 describes the procedure to get the values. The output data of the accelerometer is shown in the following chart (ref Figure Output data of the accelerometer). The offset value is the result of the difference between zero and the measured value zero point.

Figure 9: Output data of the accelerometer

x

y

z

MAX

MIN

0

249

-13

-276

-264

-3

264

280

-227

26

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

Figure 10: Output data minus offset value

x

y

z

MAX

MIN

0

262

0

-263

-267

0

267

254

-253

0

### Calibration of Gyro meter

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

## Logging

Logging is an important part when looking for debugging. The target of the logging on the NIOS II system is to provide all useful data to the developer, additionally it gives the opportunity to display those information’s nicely while flying.   
However, as task on the system the logging it has a very low priority because the logging should not use calculation power which actually would be needed by the flight controlling. So the Logging Task is running as a filler between the more important tasks.   
The data which needs to be logged are produced in the main controlling loop so the end of such a loop is the perfect place to start the logging procedure. But the Logging needs to be encapsulated from the control loop. In the controlling 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 running and reading data out of the queue and sends it to the ARM/Linux system via MCAPI

# Challenges and problems

## 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 was 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 of course 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.

What also was a problem, was that no team member really knew, how to design and structure a software system with a medium 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, what 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 the 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 essentially 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, a 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 took 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 devolved, 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 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 worked well. Whenever another user logs in to the PC, GIT gets reset and the repository have 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 PC, 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.

## Scrum

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

## Long term calendar management

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

## Small tasks take more time than thought

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

# Lesson Learned //TODO 100 mal zu wenig

# How-To Section

## ARM / Linux: VM, Cross compile, execute code

### 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:

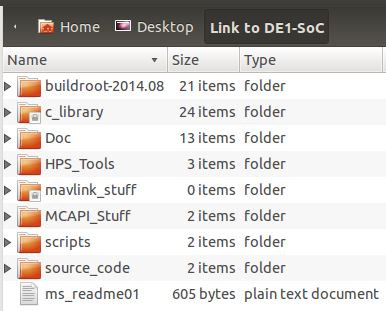
*C:\USERS\Public\QuadCopter\Ubuntu\_1204\_64bit\_de1soc\_v03*

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:



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

### 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/*

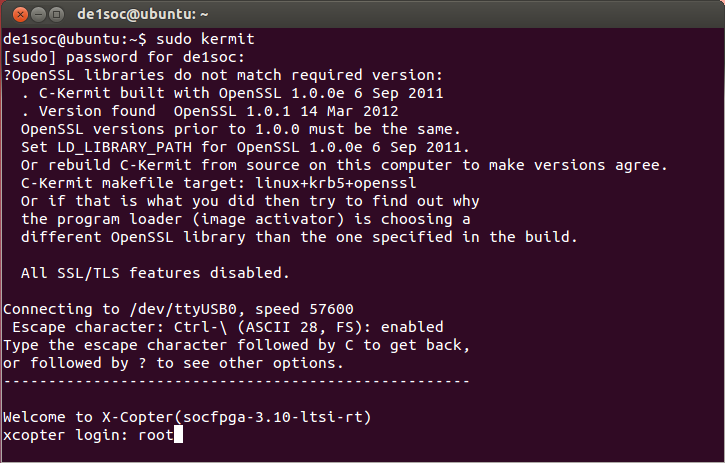
You can load your executable onto the ARM-system via SCP.

### 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”

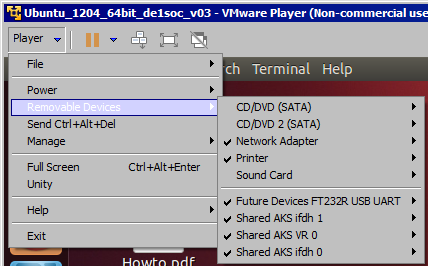
If the connection is established successfully, the terminal should look like this:



#### Problems with the connection

Hints: if the Terminal looks like this:  


* 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



## Get 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.

1. Place the rectangular form in such a way that the axis of the sensor is in the maximum position.
2. Place the rectangular form in such a way that the axis of the sensor is in the minimum position.
3. Place the rectangular form in such a way that the axis of the sensor is in the zero-point position.
4. Repeat step 4 to 6 for each axis (x, y, z).

## 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 |

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 | **4.2**V | **4.2**V |  |
| Total-Voltage | **25**V | **25**V |  |
| **Min.** | Cell-Voltage | 3V | 3V |  |
| Total-Voltage | 18V | 18V |  |
| **Max. Charge Current** | | 4.9 A (1C) | 20 A(4C) |  |

**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.

### Charging Batteries:

**Current charged per charging session** = current \* charging time

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

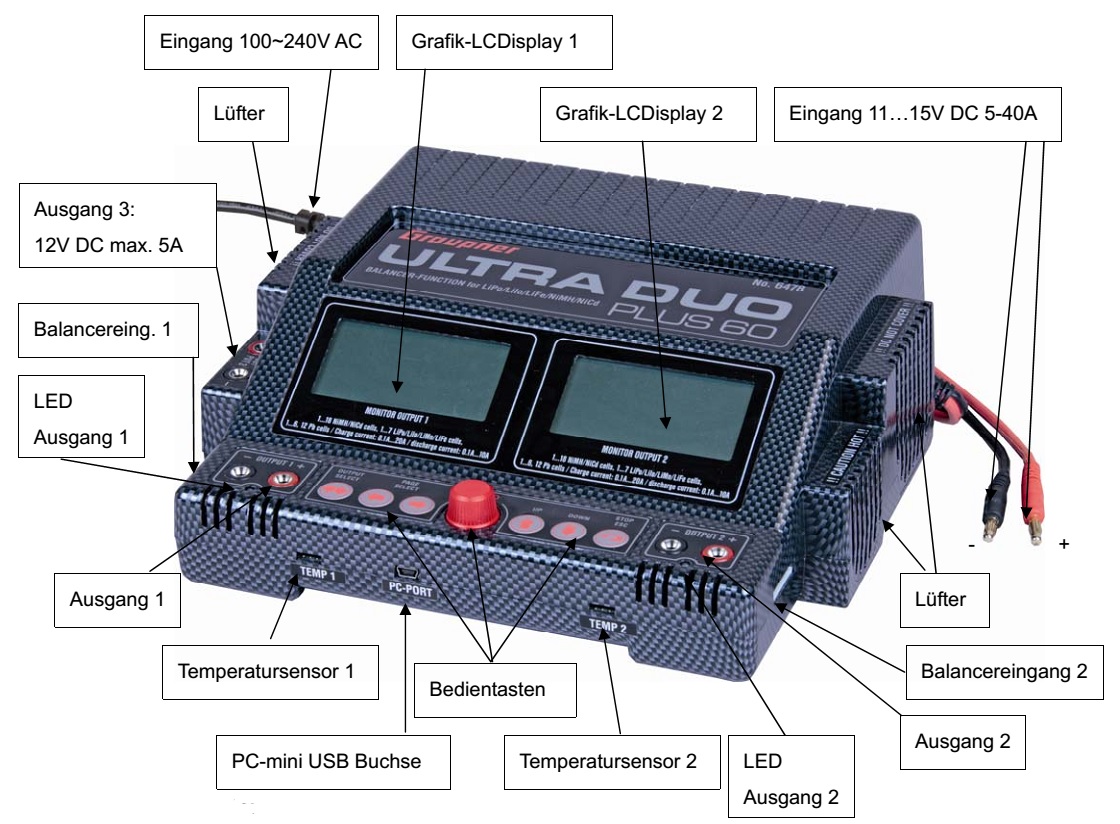
**Standard-charging-current** is 1/10 of the capacity (1.7Ah capacity -> 170mA standard-charging-current)

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

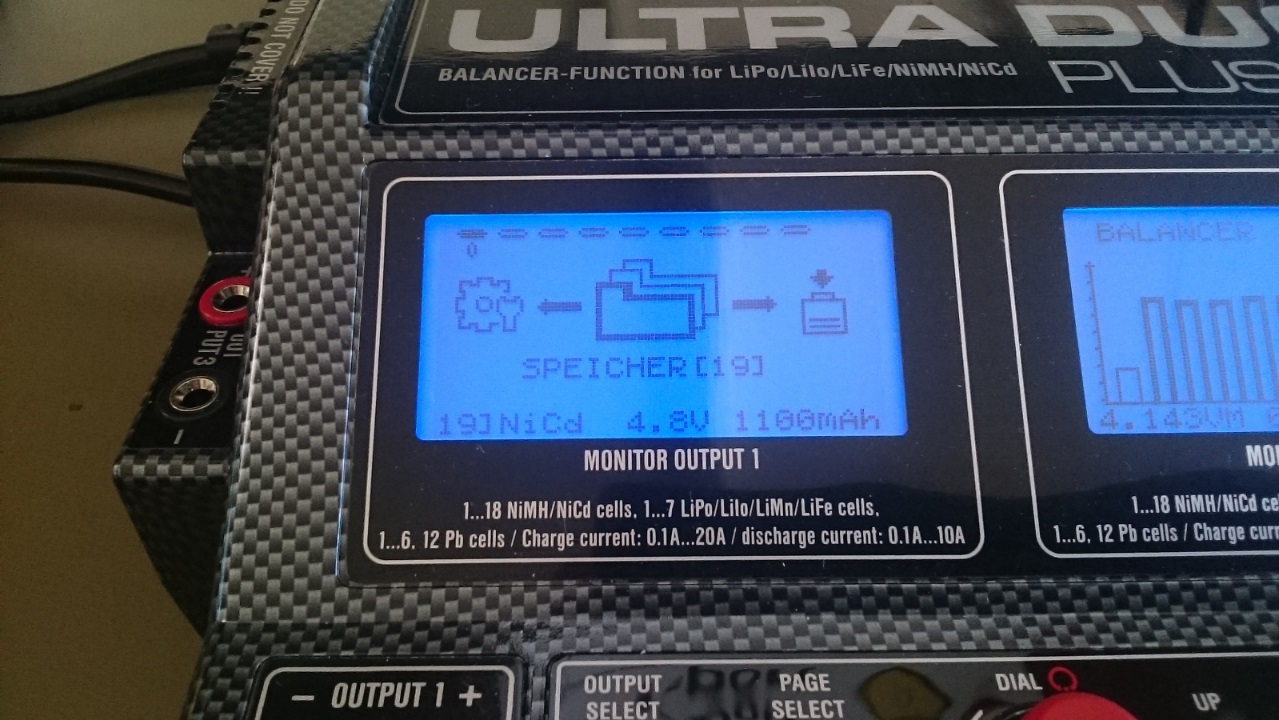
### Step-by-Step-Solution

Overview

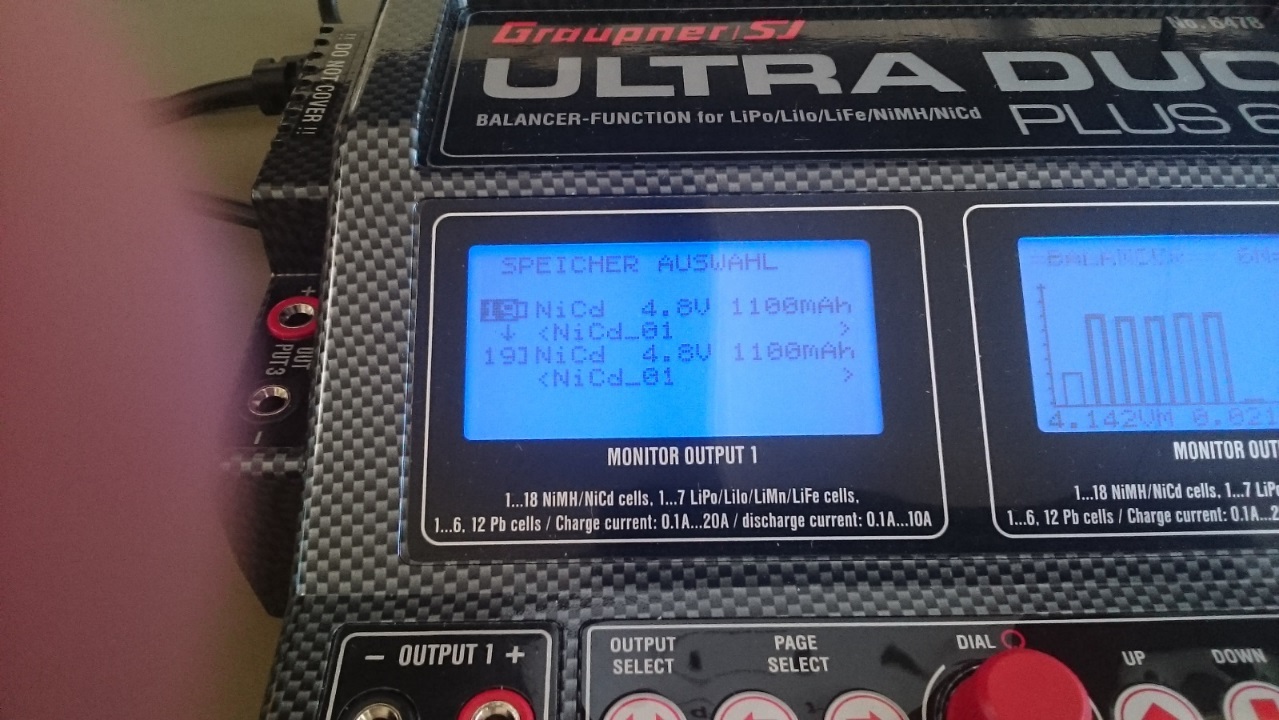
Control Interface



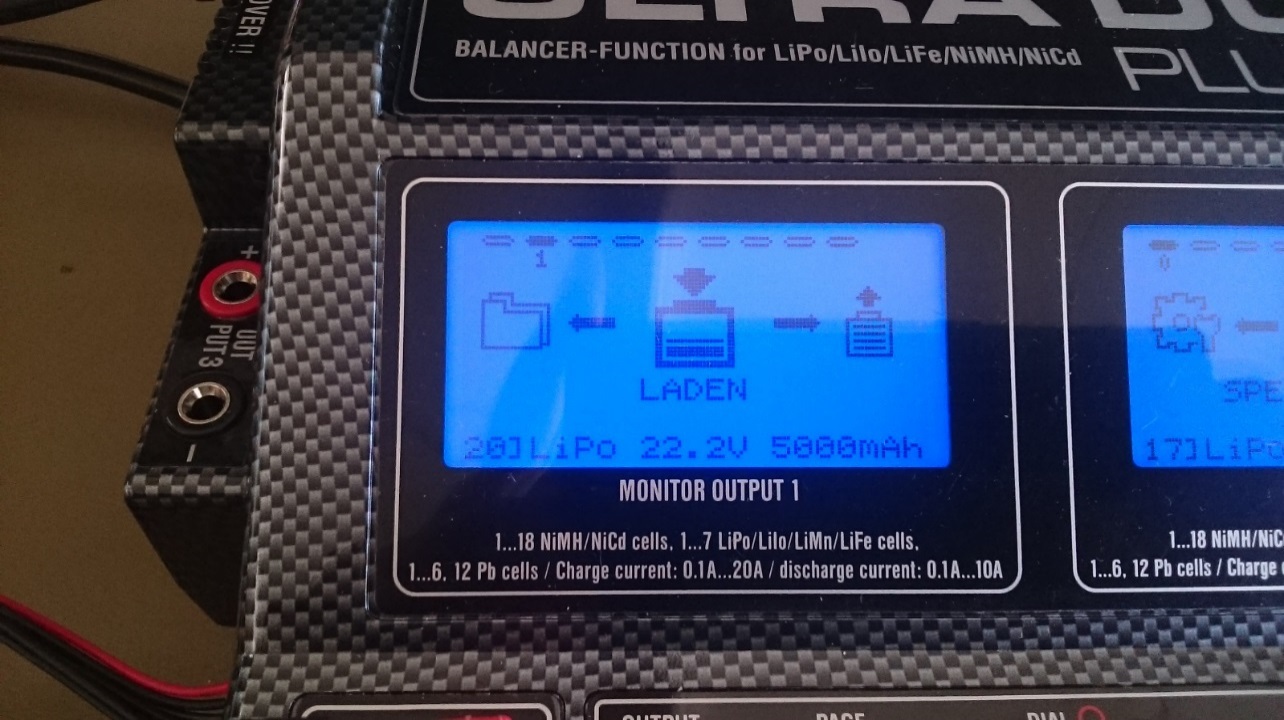
Enter “SPEICHER”-item: turn red click wheel (RCW) to select “SPEICHER” and push RCW



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



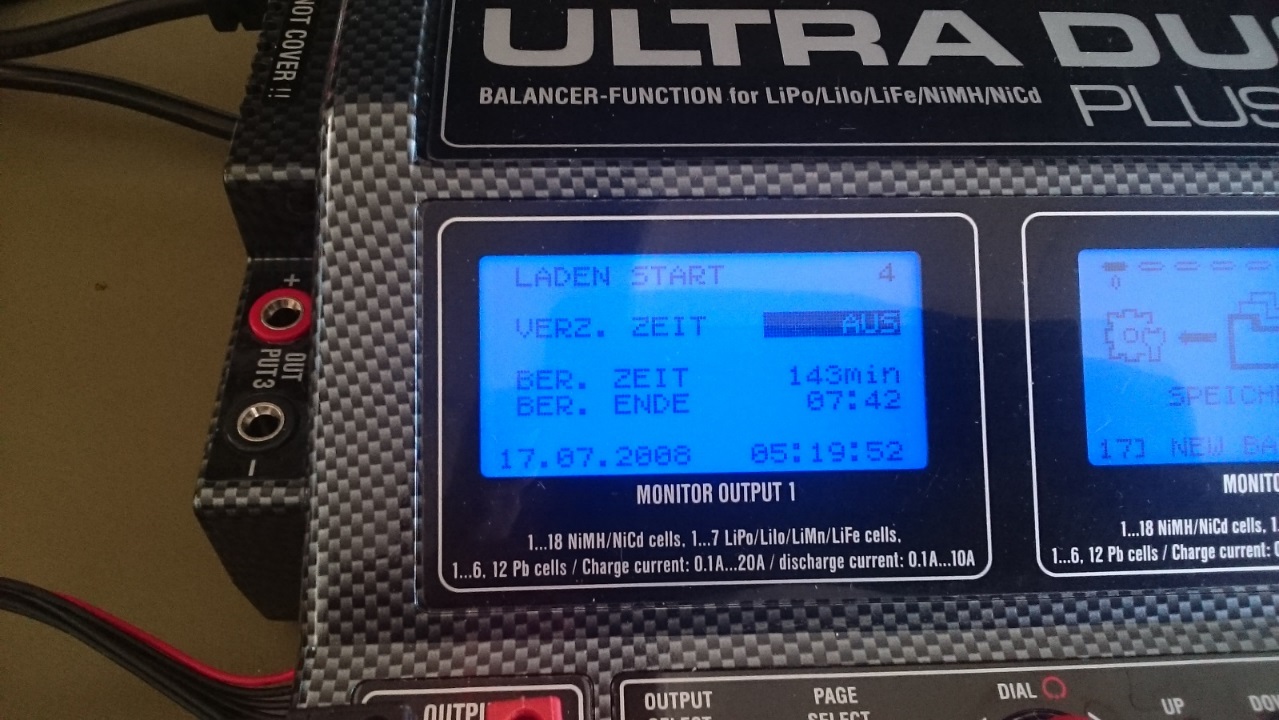
Choose Function: “LADEN” or “ENTLADEN” with RCW (here you can see the charging mode)



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



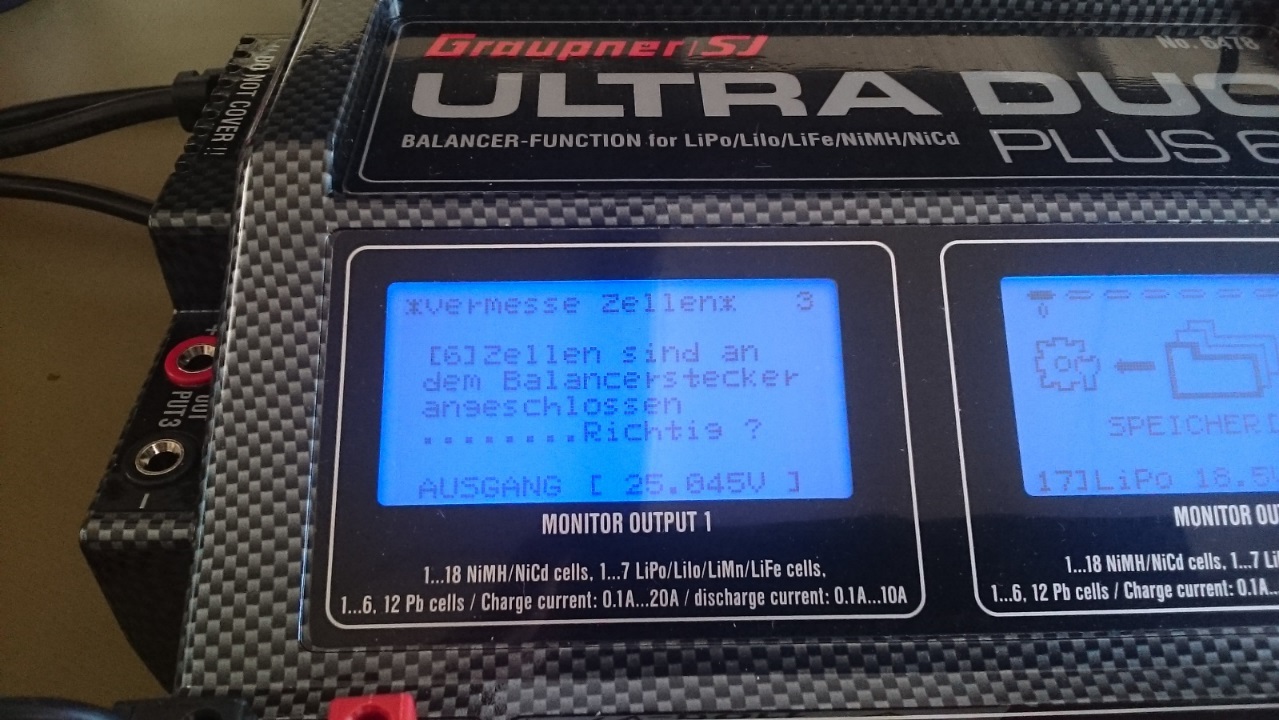
Set delay time to (VERZ. ZEIT) “AUS” (off)



Wait until connection is checked



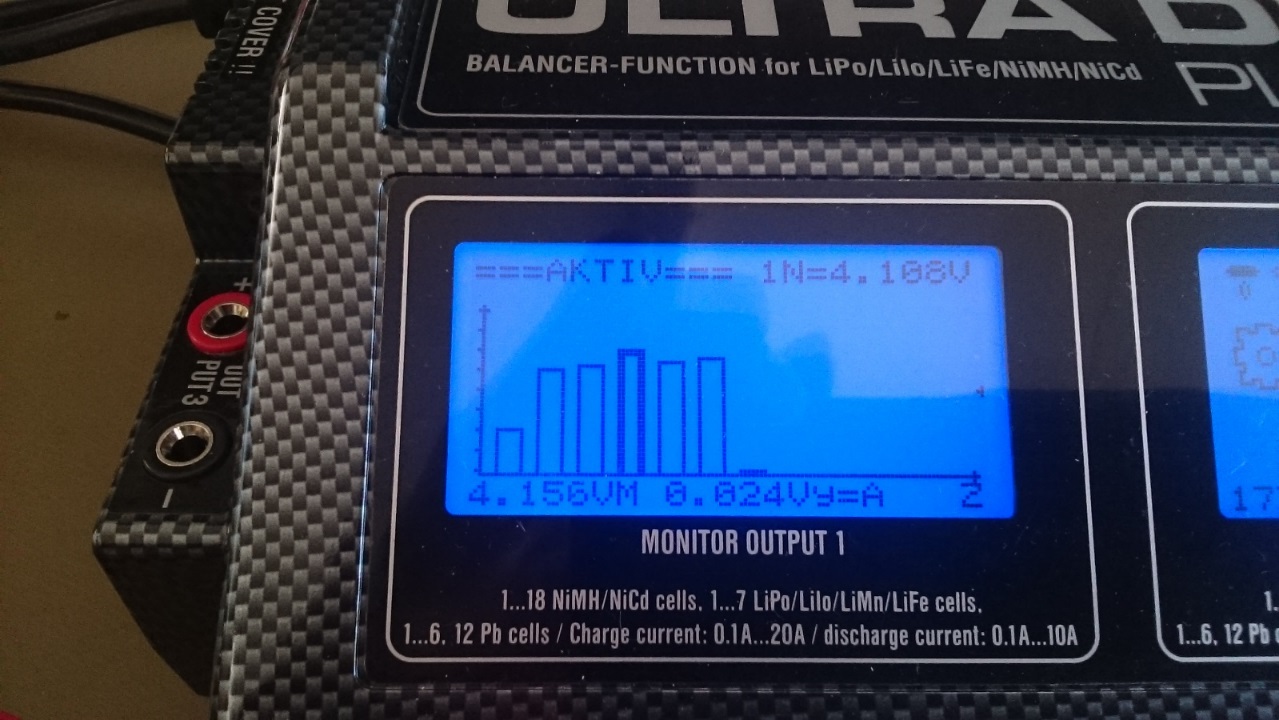
If Balance Board is connected: check cell count and confirm



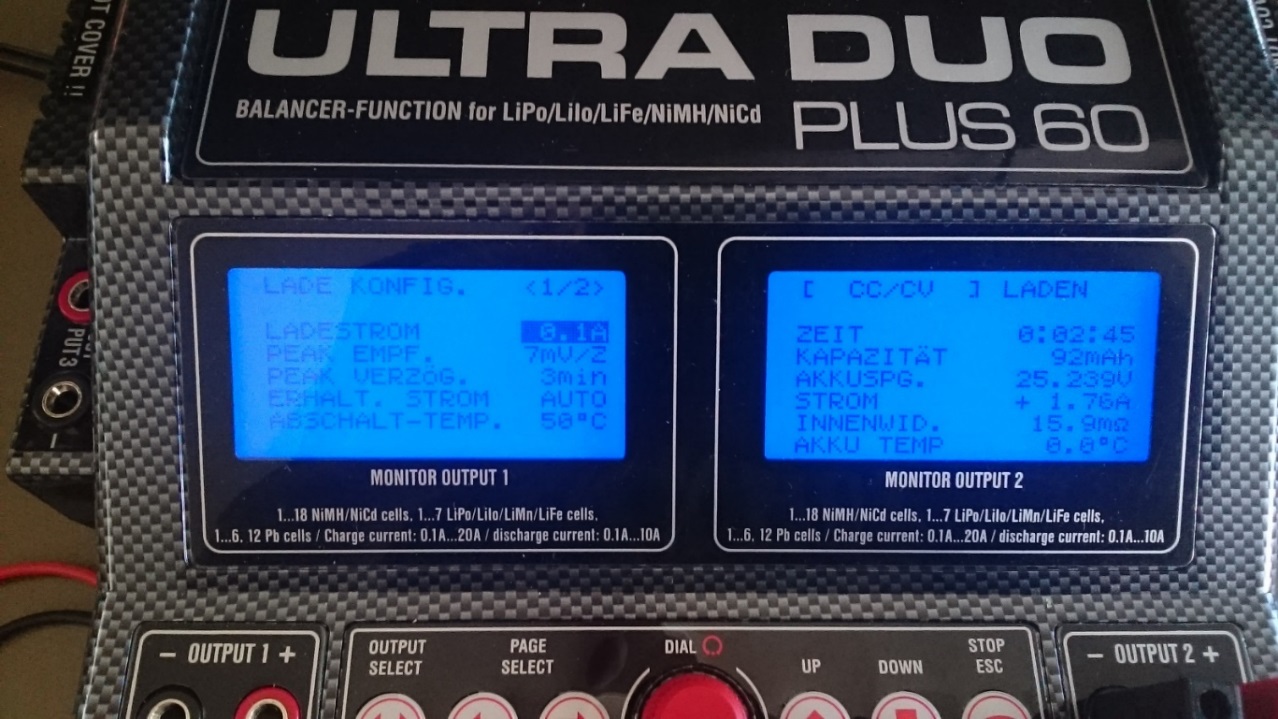
The power station starts charging and shows you current values



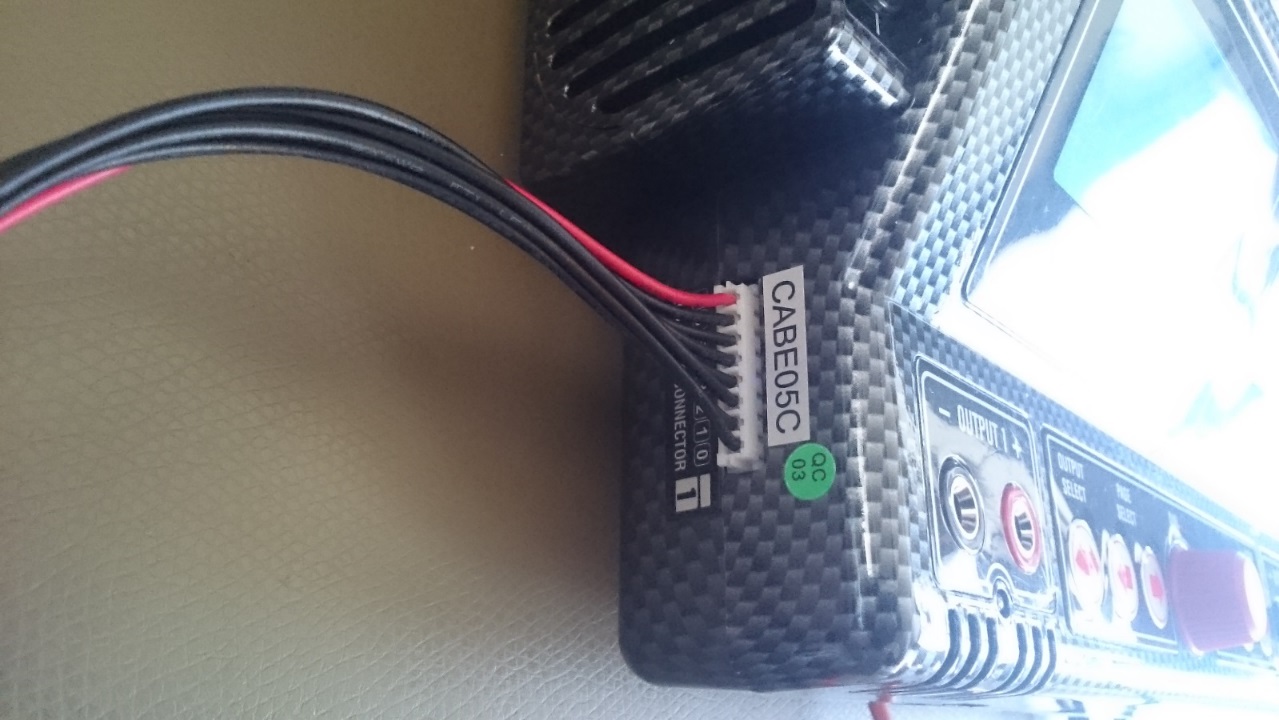
Turn the RCW to see more information



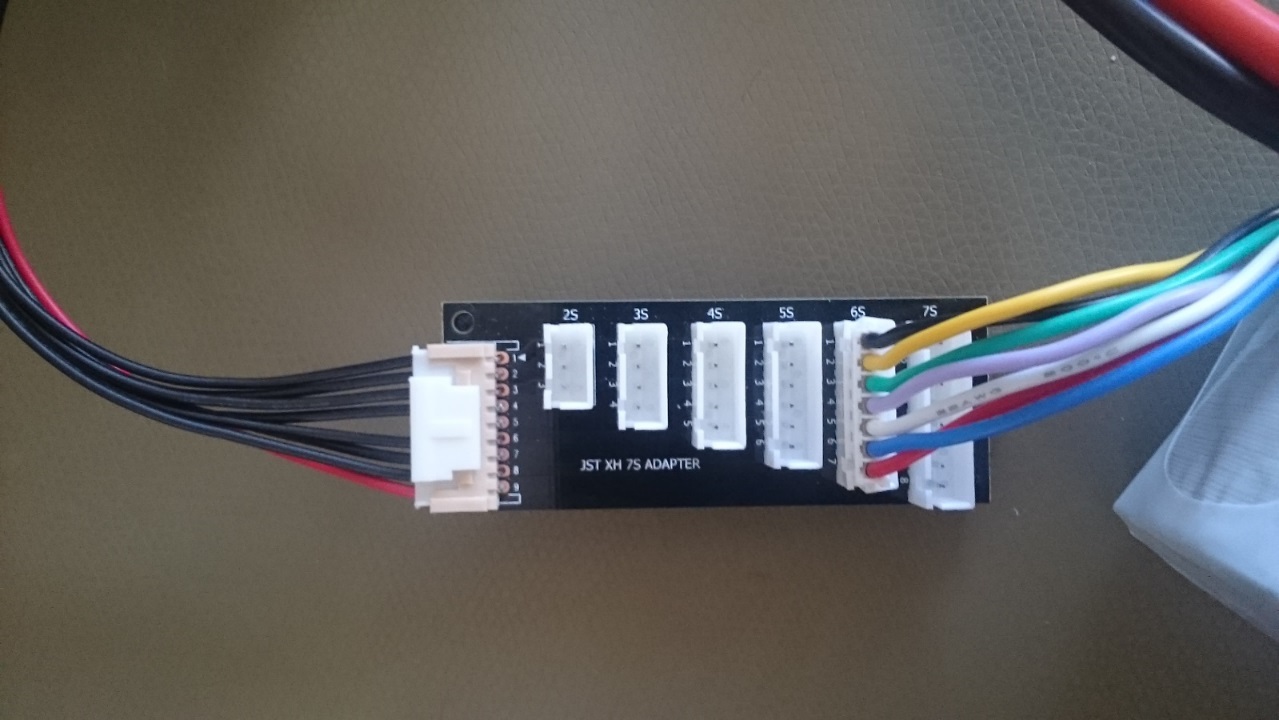
To configure the charge options press and hold RCW on “LADEN”-item until you enter der “LADE KONFIG.” settings



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



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



# Literaturverzeichnis

|  |  |
| --- | --- |
| [1] | C. D. H. S. Y. C.Löser, „Project BumbleBee,“ \\documentations\Bumblebee\_Documentation\Final\_Report.pdf, 2015. |
| [2] | Graupner GmbH & Co KG, „www.graupner.de,“ [Online]. Available: http://www.graupner.de/fileadmin/downloadcenter/anleitungen/20070222092922\_4701\_GB.pdf. [Zugriff am 11 1 2016]. |
| [3] | Microsoft, „Microsoft Developer Network,“ [Online]. Available: msdn.microsoft.com. [Zugriff am 11 1 2016]. |
| [4] | uClib.org, „Buildroot,“ [Online]. Available: https://buildroot.uclibc.org/. [Zugriff am 7 1 2016]. |
| [5] | M. H. Andreas Klinger, „Embedded-Linux-Systeme mit Buildroot erstellen,“ [Online]. Available: http://www.elektronikpraxis.vogel.de/themen/embeddedsoftwareengineering/implementierung/articles/173855/. [Zugriff am 7 1 2016]. |
| [6] | qgroundcontrol.org, „qgroundcontrol.org,“ [Online]. Available: http://qgroundcontrol.org/dev/mavlink\_linux\_integration\_tutorial. [Zugriff am 10 1 2016]. |
| [7] | D. Brunner. [Online]. Available: http://www.technik-consulting.eu/Optimierung/Quadrocopter\_PID-Regelung.html. [Zugriff am 11 1 2016]. |
| [8] | Analog Devices , „www.sparkfun.com,“ [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf. [Zugriff am 7 1 2016]. |
| [9] | Honeywell, „www.sparkfun.com,“ [Online]. Available: http://cdn.sparkfun.com/datasheets/Sensors/Magneto/HMC5883L-FDS.pdf. [Zugriff am 7 1 2016]. |
| [10] | InvenSense, „www.sparkfun.com,“ [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Gyro/PS-ITG-3200-00-01.4.pdf. [Zugriff am 7 1 2016]. |
| [11] | Graupner GmbH & Co KG, „graupner.de,“ [Online]. Available: http://www.graupner.de/mediaroot/files/6478\_ULTRA%20DUO%20PLUS%2060\_en.pdf. [Zugriff am 5 1 2016]. |